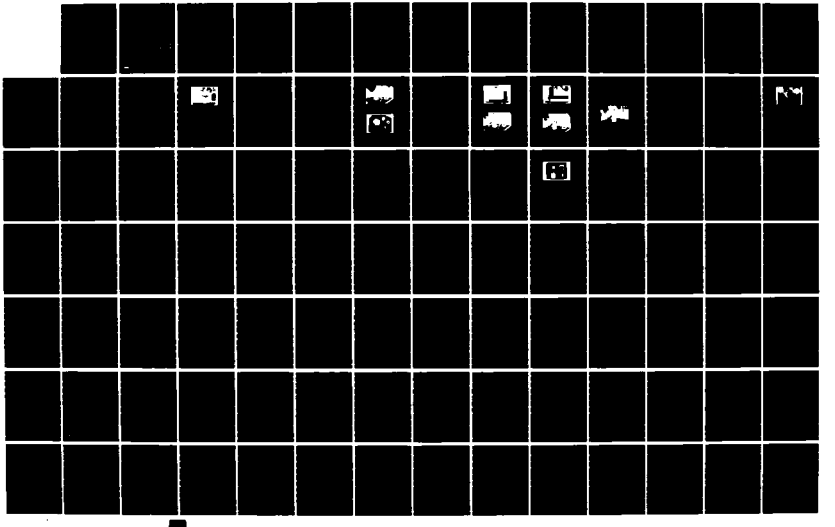
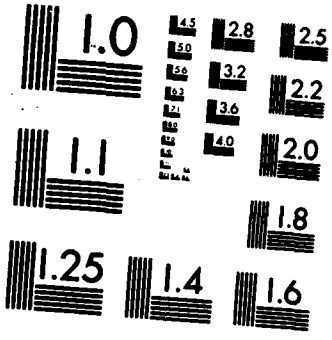
AD-A173 121 DEVELOPMENT SYSTEM FOR FLEXIBLE ASSEMBLY SYSTEM(U) SRI INTERNATIONAL MENLO PARK CA R C SMITH ET AL. AUG 86 AFOSR-TR-86-1001 F49620-84-K-0007 1/2

UNCLASSIFIED F/G 9/2 NL





(2)

DEVELOPMENT SYSTEM FOR FLEXIBLE ASSEMBLY SYSTEM

Final Report

Covering the Period 1 February 1984 through 1 June 1986

August 1986

By: Randall C. Smith, Peter Cheeseman, John K. Myers,
David Nitzan, Aviv Bergman, Antony J. Sword, and Eitan Zeiler
Robotics Laboratory

Prepared for:

Air Force Office of Scientific Research
Bolling Air Force Base, Washington, D.C. 20332

and

Air Force Wright Aeronautical Laboratories
Materials Laboratory
Manufacturing Technology Division
Wright-Patterson Air Force Base, Ohio 45433

Attn: Ted Brandewie, AFWAL/MLTC

Contract F49620-84-K-0007

SRI Project 7239

SRI International
333 Ravenswood Avenue
Menlo Park, California 94025-34393
(415)326-6200
Telex: 334486

SECURITY CLASSIFICATION OF THIS PAGE

AD-A173121

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) SRI Project 7239	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 86-1001
6a. NAME OF PERFORMING ORGANIZATION SRI International 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION U.S. Air Force
6c. ADDRESS (City, State and ZIP Code) 333 Ravenswood Avenue Menlo Park, California 94025	7b. ADDRESS (City, State and ZIP Code) Office of Scientific Research Building 410 Bolling Air Force Base, DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U.S. Air Force 8b. OFFICE SYMBOL (If applicable) NE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-84-K-0007
8c. ADDRESS (City, State and ZIP Code) Office of Scientific Research Building 410 Bolling Air Force Base, DC 20332-6448	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: 61102F PROJECT NO.: 2306 TASK NO.: A1 WORK UNIT NO.:
11. TITLE (Include Security Classification) Development System for Flexible Assembly Systems	
12. PERSONAL AUTHOR(S) Smith, R., Cheeseman, P., Myers, J., Nitzan, D., Bergman, A., Sword, A., and Zeiler, E.	
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM 2-1-84 TO 6-1-86
14. DATE OF REPORT (Yr., Mo., Day) 29 August 1986	15. PAGE COUNT 120
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

See ABSTRACT (Next page)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code) 22c. OFFICE SYMBOL

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

ABSTRACT

This final project report summarizes our research in some basic problems that make programming of flexible robotic assembly tasks difficult. The report describes our initial efforts of implementing and using an interactive graphic off-line programming system, called WORKMATE. The research results, described in five technical papers, include:

- *Locational Uncertainty.* We developed a powerful and efficient method for estimating compounded and reduced locational uncertainties among sensors, robots, and objects based on their individual uncertain spatial relationships. This method is useful in determining if the spatial errors in a planned task sequence will fall within acceptable tolerances.

- *Spatial Reasoning.* A fast algorithm, using VLSI clipping hardware in our Silicon Graphics IRIS graphics workstation, was developed to determine if a simulated moving manipulator will collide with modeled three-dimensional objects during its motions. The method is implemented in WORKMATE, which can graphically indicate the piece of the manipulator "penetrating" any of the objects at an animation speed of 4 to 6 pictures per second.

- *Sensor Usage.* Two methods were developed, using different sensor modalities, to estimate the grasping error when a manipulator hand picks up an object. This error can be compensated by proper modification of the manipulator's motion. In the first method, a camera views the hand and the object it holds, and compares their image to a "model" grasp one. In the second method, a wrist force/torque sensor measures the moments of the object during transfer motions and, based on a mass model of the object, the object location relative to the hand is estimated.

An experiment in off-line assembly programming, using WORKMATE, is described. An assembly task, requiring sensory feedback, was programmed entirely off-line, and the resulting program was subsequently executed on our real robotics testbed.

CONTENTS

ILLUSTRATIONS

I OBJECTIVE

The objective of this research is to investigate some selected basic problems in flexible assembly that make human programming of assembly tasks difficult and to incorporate the results of such investigation into a graphics-oriented, assembly task development system.

II AIR FORCE RELEVANCE

This research addresses some problems in flexible assembly of electromechanical components. Improved automation of small batch assemblies should raise production efficiency, improve product quality, and lower costs.

III RESEARCH DELIVERABLES

The bulk of our work is described in technical papers written under this contract, and delivered to the sponsor in annual reports, or as part of this document. These papers are first listed below. Following that, our research in these areas is summarized.

The following six papers have been written under this contract:

- In the research area of Locational Uncertainty:
  - "On the Representation and Estimation of Spatial Uncertainty," by R.C. Smith and P. Cheeseman; accepted for publication by the International Journal of Robotics Research.*

---

*Included in this project's Annual Report of February 1985.

- "Estimating Uncertain Spatial Relationships in Robotics," by R.C. Smith, M. Self, and P. Cheeseman; accepted for publication in the proceedings of the workshop, ***Uncertainty in Artificial Intelligence***, to be held in Philadelphia, Pennsylvania, August 1986.

- In the research area of Spatial Reasoning:

  - "Fast Robot Collision Detection Using Graphics Hardware," by R.C. Smith; published in the Proceedings of the Symposium on Robot Control (SYROCO), Barcelona, Spain, 1985.*

- In the research area of Sensor Usage:

  - "Determining An Object's Location in a Robot's Hand By Means of Vision," by Eitan Zeiler, Robotics Laboratory Technical Note, SRI International, August 1984.

  - "Estimating Object Location in a Manipulator's Hand Using Force/Torque Information," by A. Bergman and R.C. Smith, to be submitted for publication.

- In the research area of Off-Line Programming:

  - "Robot WORKMATE: Interactive-Graphic Off-Line Programming," by R.C. Smith, Robotics Laboratory Technical Note, SRI International, February 1986.†

---

*Included in this project's Annual Report of February 1985.

†The material in this technical note was included in the body of this project's Annual Report of February 1985.

IV THE DIFFICULTIES IN MANUAL PROGRAMMING

It is difficult to write programs for a flexible assembly system because of the sensing and decision making capabilities entailed. From previous experiences at SRI in sensor-guided assembly, we picked three related problems we considered most important for research:

- *Locational Uncertainty*---Estimation of errors in relations among workpieces, sensors, and effectors due to part tolerances, measurement errors, and positioning errors.

- *Spatial Reasoning*---Analyzing the relationships among solid objects in a three-dimensional space.

- *Sensor Usage*---Selecting sensors, determining their parameters, estimating sensor output values, and verifying actions by sensing (execution monitoring).

In addition to these topics, research in off-line robot programming was performed, and an inter-active graphic robot programming and simulation system was developed, called WORKMATE.

A. Location Uncertainty

In many applications of robotics, such as industrial automation and mobile robots, there is a need to represent and reason about spatial uncertainty. In the past, this need has been circumvented by special purpose methods such as precision engineering, very accurate sensors and the use of fixtures and calibration points. While these methods sometimes supply sufficient accuracy to avoid the need to represent uncertainty explicitly, they are usually costly. An alternative approach is to use multiple, overlapping, lower resolution sensors and effectors and to combine all the spatial information (including the uncertainty) from all sources to obtain the best spatial estimate. This integrated information can often supply sufficient accuracy to avoid the need for the hard engineered approach.

In addition to lower hardware cost, the explicit estimation of uncertain spatial information makes it possible to decide in advance whether proposed operations are likely to fail because of accumulated uncertainty, and whether proposed sensor information will be sufficient to reduce the uncertainty to tolerable limits.

A difficulty in combining uncertain spatial information is that it often occurs in the form of uncertain relative information. This is particularly true where many different frames of reference are used, and the uncertain spatial information relative to yet another frame is required. Our approach presents a general solution to the problem of estimating uncertain spatial relationships, regardless of which frame the information is presented in, or in which frame the answer is required. The basic theory assumes that the errors are "small," so that the nonlinear transformations from one frame to another are approximately linear.

A representation of spatial relationships which incorporates knowledge about uncertainties, in the form of probabilities, was described in the previous report and paper. It was a significant advance over previous work which generally handled uncertainties only by worst-case analysis, and was thus very conservative. Our initial research concentrated on spatial relationships with three degrees of freedom (two translations and a rotation in the plane). Within that framework, we presented a first-order method for estimating the error when uncertain relationships were

(1) Compounded, increasing the overall uncertainty

(2) Merged, or "averaged," reducing the uncertainty.

Compounding is illustrated by the sequential motions of a mobile robot, whose every (noisy) move increases the uncertainty about its current location with reference to its starting point.

In merging, two estimates of a relationship are combined to produce a better estimate of the relationship. A (noisy) sensor measurement of the robot's location

can be combined with a second estimate of the location obtained by compounding the movements, providing a better location estimate than either of the original pieces of information.

Appended to this report is a new paper describing the final results of our research on this topic. We have integrated the approach described in the previous paper, with the formalisms of recursive estimation theory, which provides a solid theoretical framework, and points the way to numerous extensions.

In the new paper, uncertain spatial relationships are tied together in a representation called the "stochastic map." It contains estimates of the spatial relationships, their uncertainties, and their inter-dependencies. The paper describes the map structure, followed by methods for extracting information from it. Finally, a procedure is given for building the map "incrementally," as new spatial information is obtained. The map contains the first-order estimate of the mean and covariance of the uncertain relationships described, using all the available information. In addition, general constraints on the spatial variables can be specified, such as colinearity, or coplanarity of points, and the information in the map will be updated accordingly. The constraints may even be given with tolerances; i.e., the constraints can be stochastic, rather than absolute. The paper illustrates a simple "rectangular" constraint on four poorly-known spatial points, with a resultant large decrease in their uncertainties.

The theory is illustrated by an example of a mobile robot acquiring knowledge about its location and the organization of its environment by sensing at different times and in different places. The theory is readily extended to six degrees of freedom, and the formulae for this extension are given in the paper's appendix. Ultimately, we believe our results are applicable to off-line planning of sensor and manipulation strategies in numerous robotic domains.

B. Spatial Reasoning

A particularly important problem in spatial reasoning is the problem of collision avoidance---finding a safe trajectory for a manipulator through an environment of obstacles. The general problem is unsolved. Our work has focused on developing tools that aid a user of an off-line robot programming system in defining robot trajectories that are collision free. Robot motions can be visually inspected by the programmer, if they are graphically simulated. However, experimentation showed that this kind of visual inspection for collisions in the simulated robot workcell was tedious and prone to error. An automatic technique to detect simulated collisions quickly was developed, and described in a previous paper appended to an annual report. The technique relies on the use of VLSI "clipping" hardware, which will be common in advanced graphic workstations of the future. Such hardware exists in the IRIS 2400 graphic workstation used in our off-line programming system, called WORKMATE. The algorithm can detect collisions (in simulation) between a manipulator and its environment at high speed---sufficiently fast for animation. It was implemented in WORKMATE.

C. Sensor Usage

It is highly desirable to develop approaches for determining the location of a workpiece in a manipulator's hand; the error in the grasp can then be estimated and corrective motions made by the manipulator. Two methods were developed to determine the error in the grasp: one, by viewing the hand-held object with a camera; the other, by measuring forces and torques exerted by the object on the robot's wrist.

1. *Grasp Correction with Vision*

In the first approach, a prototype of the object is grasped by the manipulator, and brought to an inspection station, where a visual prototype of the object, when correctly grasped, will be trained. The manipulator presents the object to the camera several times, each time positioning its gripper at the same location under the camera. A binary image of the correctly grasped object is first obtained, and image features of the object in this "model" grasp are stored. In subsequent training steps, the object's location in the hand is perturbed, in one degree-of-freedom at a time. Since the manipulator always positions its hand at the same location, the perturbed object will appear to have moved in the image, with respect to the model image. Image features of the object are extracted, and associated with the known error magnitude, and the spatial degree-of-freedom in which the error is introduced. The statistics over a number of such training steps can be used to build a sensitivity matrix, which relates the magnitudes of the given errors in the spatial dimensions, to the magnitudes of changes in the image features of the object.

After training, the manipulator can now acquire an object, with an unknown grasping error, and bring the object to the inspection station. Features from the image of the object are compared to the features stored for the object when it was held correctly. By using the inverse of the sensitivity matrix, developed during the training procedure, it is possible to estimate the grasping error based on the disparity of the image features. The method, its implementation, and experiments are described in an appended paper.

2. *Grasp Correction by Force/Torque Sensing*

In the second approach, a force/torque sensor mounted on the manipulator's wrist is used to determine the grasping error. The position of the object in the gripper is determined by measuring forces and torques at the robot wrist after putting the gripper in several different positions where the robot is either at rest or moving in constant velocity; the orientation error of the part in the gripper is determined by measuring forces and torques at the robot wrist during a controlled acceleration.

The procedure is practical to implement, and it is theoretically feasible to accurately estimate the error in the grasp of a held object, even while the manipulator acquires and transfers it. One advantage of this procedure, compared with the previously described method, lies in removing the necessity of first moving the part to an inspection area, where the grasping error would be determine---thus saving time. A major implementation problem is that robot manufacturers do not currently provide the user with velocity and acceleration control over the robot. However, rather than controlling the acceleration of the robot, we can define several fixed motions for the robot to make, and measure the accelerations. When these motions are later made by the manipulator, the acceleration parameter, estimated a priori, can be used in our calculations. The detailed theory behind this method is presented in an appended paper.

D. Off-Line Programming With WORKMATE

An off-line robot programming system, called WORKMATE, was developed under this contract and demonstrated to the Industrial Affiliates of the Robotics Laboratory at SRI, and to the sponsor. Its description has been submitted as part of an annual report. The work has been extended, and a further off-line programming experiment has been performed. A robot assembly task was written off-line, then executed on the real robotic testbed. The programs generated included the use of sensory feedback to determine part locations that were unknown before on-line execution. The results, described below (Section V), detail the deficiencies

and virtues of the current implementation, so that future research can be planned appropriately.

V OFF-LINE PROGRAMMING USING WORKMATE

A. Overview

1. *Assembly Station Configuration*

The SRI Assembly Testbed Station consists of two PUMA 560 robots, controlled by VAL-II, and an Automatix AV-4 vision module. Each of these application modules is controlled by a dedicated LSI-11. The LSI-11 module computers are controlled in a hierarchical fashion by a VAX-11/750 computer. In addition, there is a Silicon Graphics IRIS-2000 workstation connected to the VAX.

2. *WORKMATE Description*

WORKMATE is an off-line programming system developed for the IRIS by Randall Smith of SRI International under AFOSR Contract F-49620-84-K-0007P00001. The WORKMATE acronym stands for WORKstation Modeling, Analysis, Training, and Emulation. The system is able to model robots, fixtures, and parts, using a polygon-based surface modeling package. Backwards arm solutions and relative positionings are included in the package; the ability to servo the robots requires a separate (forward) arm solution package that must be built for each type of robot modeled. Figure 1 shows the model of the assembly station used in this experiment.

The system gives a true-perspective three-dimensional shaded polygon representation of the scene. Options include wire-frame depiction, and red-green stereo for filled-polygon or wire-frame modes. Shading is based on a single light-source at infinity, with self-shading only; no shadows are generated. Scenes are generated at four frames per second. The viewpoint is controlled by the mouse. The current in-house hidden-surface algorithm requires special coding for intra-object surface ordering as part of the modeling process. Because the polygons in each object have a consistent relationship with each other, they can be ordered

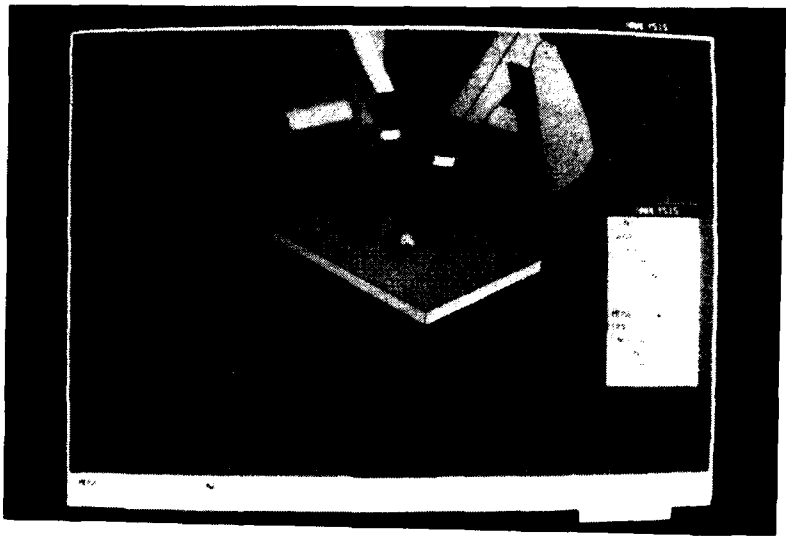


Figure 1: WORKMATE Model of the Assembly Station

into a display tree at compile time. However, since objects can be moved, the system does not attempt to make a viewpoint-dependent object tree. Instead, the objects are displayed in a fixed order, regardless of the viewpoint. This is seen as one of the major areas needing improvement; preferably, special-purpose hardware would eliminate the need for such labor.

The WORKMATE system has three main robot-connected capabilities. The robots can be "servoed" and positioned, using various modes of motion. A robot program can be created, using "taught" positions, and the resulting program can be "played back" for examination and verification. After the operator is satisfied, the final program can be sent to a text file on disk for eventual execution.

3. *Off-Line Programming Scenario*

The name "off-line programming" describes a system that is able to generate executable programs at an arbitrary time, away from the actual robot station that will be doing the work. In other words, the programming is not done "on-line," that is, using the running equipment.

An off-line programming scenario in our laboratory starts with the operator sitting down at the IRIS graphics computer and generating a robot program using WORKMATE. The robot program is a series of commands; it is a simple text file that can be printed out or copied from one computer to another. Our programs are sent to the VAX for storage.

At a later time, the program may be executed on the actual robot station. A station-master program containing an interpreter reads in the robot program, performs the necessary calculations and command decompositions, and dispatches module-level commands to the appropriate LSI-11 module computers. The module computers perform similar calculations and decompositions which result in device-specific commands. Each of these commands is then sent to the appropriate device, which may actually contain its own microprocessor controller. Commands may be synchronous and tie up the controller until a reply is received; they may be polled, in which case the controller goes off and later checks to see if a response has been received; or, they may be of the "fire and forget" variety. Thus, both the module computers and the station supervisor computer may use completion signals in various ways. In this manner, a set of completion messages similar to the command messages flows up the computer hierarchy.

B. Assembly Experiment

1. *Assembly Parts*

To demonstrate the capability of the off-line programming system, a lantern flashlight was chosen as an example of part assembly. The flashlight selected was the Eveready Energizer Halogen Light No. 209HS, a recent model that has been designed well for ease of assembly. The flashlight consists of four parts: the case, or flashlight body; the battery; the light bulb, reflector, and switch assembly; and the cover top which surrounds and protects the light assembly. Figure 2 shows the actual parts used in the experiment; Figure 3 shows a graphical display of the part models used by the system.

a. The Case

The black flashlight case is molded out of a stiff plastic. It has a flexible button attached flush on the outside, with a chamfered interface area on the inside. When the light assembly is put on the flashlight, a cylindrical switch plunger on the assembly is inserted into the interface area. (The switch is activated by pressing an outside button, which uses a rigid extender in the interface area to depress the switch plunger.) The flashlight has a cylindrical mouth opening with an 86 mm inside diameter. Although this opening is manufactured threaded so that the light assembly screws into place after it has been inserted, for our experiment the threads on the case were shaved off, resulting in a press-fit after insertion. We felt that the experiment as carried out was sufficiently complex as to demonstrate the concept of off-line programming, without introducing difficulties that would strain the limits of both the capabilities of the robots and the time constraints of the project.

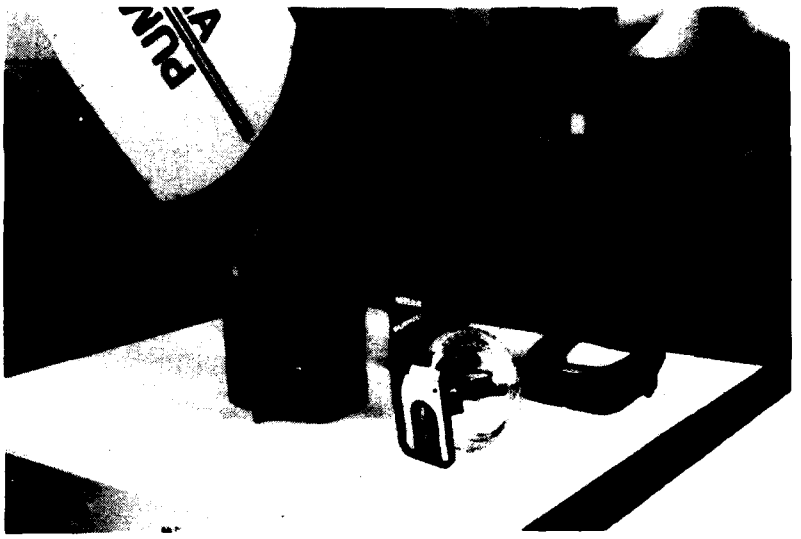


Figure 2: Assembly Parts
(Flashlight case, battery, light assembly, and cover)

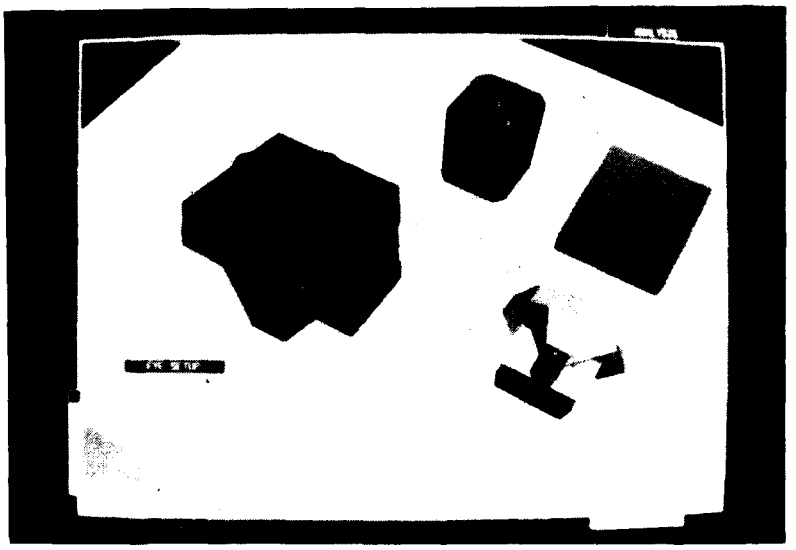


Figure 3: Models of the Assembly Parts

b. The Battery

The battery is a "super heavy duty" standard 6-volt lantern battery. It weighs 1.3 pounds, and has smooth metallic sides. The diagonal dimension of the battery is 82 mm, resulting in a +/- 2 mm tolerance for a straight insertion. The battery is the first part to be inserted into the case. Figure 4 shows the system's view of the insertion (the second robot has been deleted for ease of display). Figure 5 shows the actual insertion.

c. The Light Assembly

The light assembly consists of a parabolic reflector containing the lamp, attached to the lens in front, the switch on the top, and a contact plate in the back. The lens has a cylindrical threaded flange, 15 mm deep, that fits over the mouth of the flashlight. The switch, as mentioned before, must be oriented in the proper direction so as to engage the interface area when the light is inserted into the flashlight case. The contact plate is a rectangle 60 x 90 x 7 mm thick, which presses down against the battery springs in the assembled flashlight. Since the rectangle is too large to let the light assembly be inserted directly, the light assembly insertion requires a series of intricate movements. First, the assembly is rotated 90 degrees so that the rectangular plate and the lens flange are end-on to the flashlight mouth, as shown in Figure 6. The assembly is lowered until the end of the plate is beneath the rim of the case's mouth (see Figure 7). Then, the light assembly is rotated back towards the case, which hooks the plate under the inside of the mouth. The rotation is continued slightly and the assembly is lowered, to seat the switch in the interface area and hook the front end of the lens flange (closest to the white button) over the flashlight mouth. Finally, the assembly is lowered still further and rotated slightly backwards again, to seat the rest of the flange around the flashlight mouth.

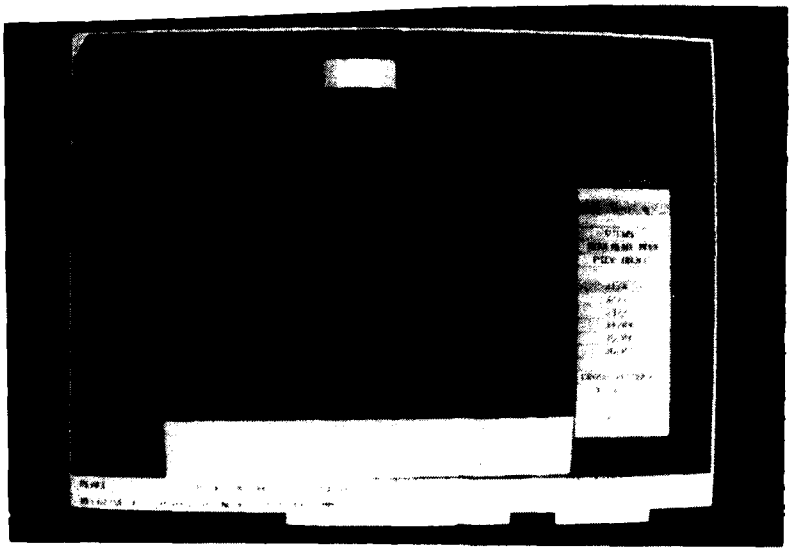


Figure 4: Off-Line Programming of the Battery Insertion

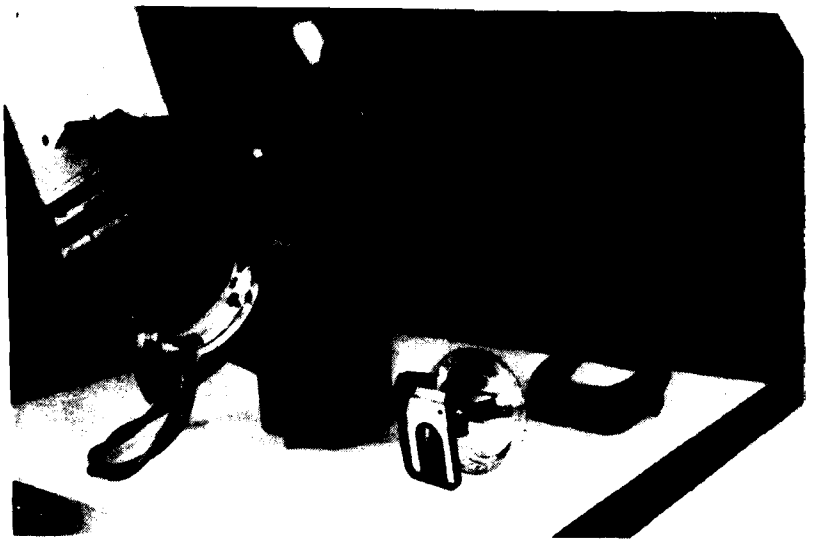


Figure 5: The Resulting Execution of the Battery Insertion

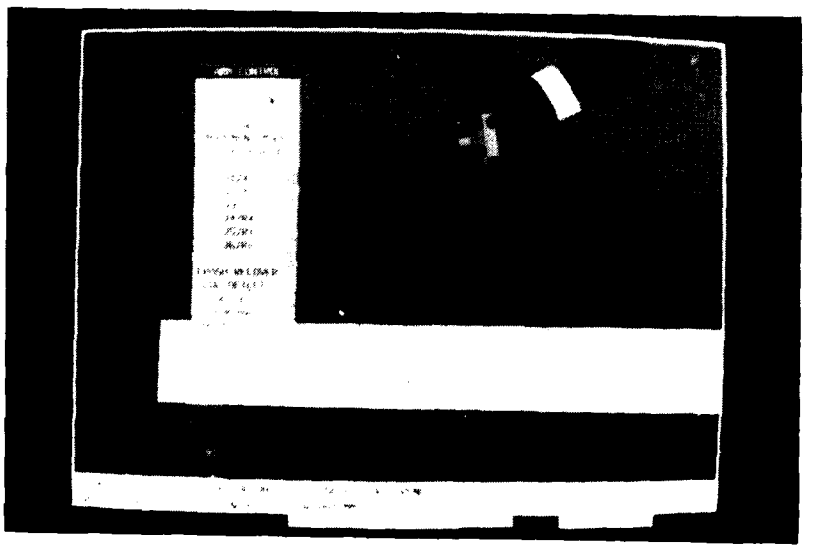


Figure 6: Off-Line Simulated Positioning of the Light Assembly Above the Flashlight Case

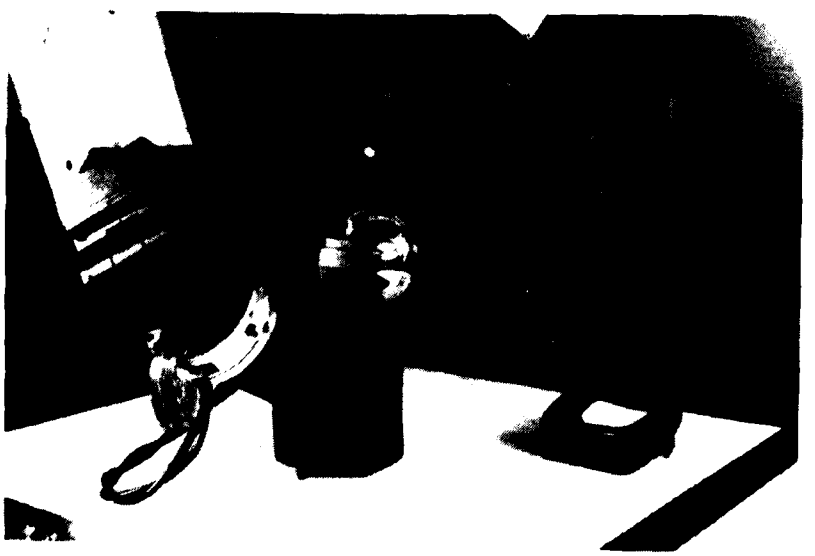


Figure 7: Execution of the Light Assembly Insertion

d. The Cover

The cover is made of a stiff, bendable black plastic. It is square, like the outside of the flashlight case. The cover fits around the lens and the flange. It has a pair of clips, or latches, that stick down on the left and right sides of the case and serve to hold it in place. These are supposed to snap onto ledge depressions when the cover has been pressed on fully with the proper alignment. Figure 8 shows the results of this step.

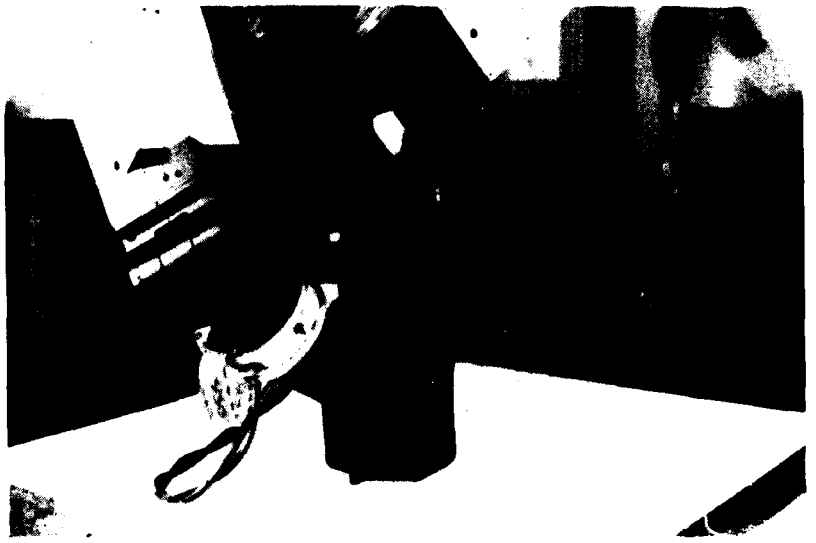


Figure 8: Assembling the Cover onto the Flashlight Subassembly

2. *Results*

This section provides an overview summary of the results obtained from this experiment.

- The ability to write programs for robots off-line, using WORKMATE, was demonstrated.
- The ability to write off-line programs entailing the use of sensory feedback to determine previously unknown conditions and locations on-line was demonstrated.
- The ability to send programs written off-line to an actual robot station, and to execute them, was demonstrated.
- The execution of the programs faithfully followed the quality of the program as designed off-line. A good program was able to successfully assemble the flashlight.
- Much was learned from the experiment. An important resulting point is that breakdowns occur when the simulation fails to model the real world appropriately.

C. Experience with the System

We experimented with different configurations for the system. Based upon our observations, we found that the ease of use of the system depends on many factors. These can be broadly classified as making the user interface elegant, controllable, and fast.

1. *Realistic and Detailed Model*

Part of the process of making the user interface elegant is providing a realistic and detailed model to the user. The hidden-surface algorithm should correctly clip the unseen parts of the polygons from all viewpoints so that occlusion cues can be employed by the user to determine the relative locations of objects in space. The parts should be shaded, depending upon their angle to the viewpoint so that the user can employ shading cues to determine spatial orientation. Both of these serve to make the scene appear more realistic.

The scene as portrayed must also be detailed, although it is acceptable to make some approximations for the sake of efficiency. Explicit details make the user feels that he understands the scene better; he instantly recognizes what is going on, as opposed to having to spend time figuring out what over-simplified blob-like models on the screen are supposed to represent. Details are also mandatory when complicated, intricate motions are programmed by the user, based on the configuration of the parts. For example, the light-assembly insertion was programmed based on inserting the rectangular plate of the light in past the rim of the flashlight case's mouth. In order to do this correctly, all the details of these parts had to be modeled accurately, as shown in Figure 9.

2. *Simulation of Arm Motion*

A useful feature of the system was the ability to simulate arm motion. Both straight-line and joint-interpolated motion were modeled. Goal positions could be specified as precision-point poses, absolute Cartesian points, or relative Cartesian points.

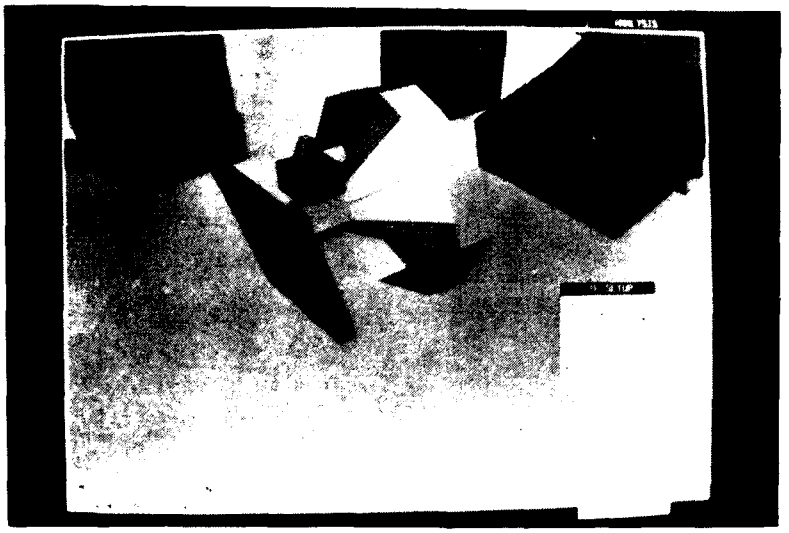


Figure 9: Closeup of the Necessary Details Used in Modeling the Light Assembly Part

3. *Movable Eye while Inside Routines*

A recent extension to WORKMATE is the ability to relocate the eye while the user is inside other movement-defining routines. It is useful to be able to move around a scene and look at a situation from several different angles, especially during the definition of a complex or intricate movement sequence.

4. *Motion and Alignment of Parts*

A very useful feature of the WORKMATE system is the ability to move the arm relative to coordinate frames attached to parts on the table (as well as the traditional world- and tool-relative motions). In addition, the system has the capability of defining motions to particular relative locations, i.e., directly above the part, or aligned with the part, or both. This capability has the advantage of specifying an entire movement sequence with just one instruction: one can specify a motion to align the arm's hand above a part to be mated, and the system immediately generates the program instruction and moves the arm to that position.

The "above" position for each part is based on a prespecified height; the "align" position projects the current location of the tool onto a prespecified axis of the part, with the tool then oriented along the axis.

5. *Real-Time Turnaround*

We found that it was very important for the system to have a real-time turnaround, i.e., more than four frames displayed each second. If the rate is lower, then not only is the illusion of motion destroyed, but also the mouse cannot be controlled properly: The user overshoots with the mouse on mouse-driven motion commands; he then corrects, but overshoots on the correction---all because the feedback is not real-time. Long turnaround time is a serious problem for both servoing the arm and moving the eye's viewpoint using the mouse.

The speed of the system depends on the number of polygons in the model, the amount of processing that has to be done to compute the display, and the processor speed. The speed also depends on whether the entire software system can be stored in the physical memory at once, or whether the system is forced to page sections of the program to disk in order to keep up the virtual memory space. We could not modify the first factors much, except work without one of the robots in the scene when we were only testing out algorithms; however, we were able to affect the last factor by cutting down the physical size of the code and by using additional memory boards. Having sufficient physical memory to keep the entire system in memory dramatically increased the turnaround time from one frame every half-minute to subsecond rates.

6. *Use of Defaults*

It is important to maintain the speed of the system by designing the user interface to require as few mouse clicks as possible. This can be done by extensive use of defaults in the system parameters.

A robot controlling system as complex as WORKMATE requires many parameters to specify the robot motion; e.g., which robot or hand is going to be moved, whether the movement is joint interpolated or straight-line, whether the goal position is a precision-point joint pose or a coordinate point, and whether the coordinate point, if used, is world relative, robot relative, or part relative. If a part-relative coordinate system is used, the appropriate part must be specified. All of these commands only define the motion as output to the final generated robot program; a similar series of parameters must be specified to define the motion used to servo a robot to the position to be taught.

The most straightforward implementation of the system requires the user to specify each of these parameters for every motion taught. Even though the specification of a single parameter requires only a single mouse click on a displayed menu, the amount of time spent clicking the mouse significantly slows the use of the system down when each parameter must be defined. A solution to this problem is to use defaults in any and all cases possible, while maintaining the power to change parameters if required. For example, the system now works with the "current robot," instead of requiring that the user specifies which robot he is working with every time another movement is taught. This provision is important because the amount of mouse clicks required to interact with the system basically defines the amount of time required to write a complete robot program. We found that such an apparently insignificant matter as the number of mouse clicks required for each movement had a large impact on system usability.

D. Discussion

1. *Successes*

a. Rigid Parts

The modeling system is at its best when it represents rigid parts. The light assembly, the battery, and the case were all represented well by the system---especially when it came to acquiring the pieces using a robot.

b. Intricate Motions

The system excelled at programming intricate motions. For instance, seven intermediate positions were successfully programmed to insert the light assembly into the case, as detailed in the light-assembly parts section. Other motions were successfully programmed to find the light, pick up the light, move out of the way, find the case, and move over the case. The movements entailed in inserting the battery and the cover employed similar series of intricate motions.

One of the advantages of the system was the controllability of the view. Since the scene can be viewed from any angle and at any degree of magnification, it is simple to enlarge the scene from a convenient angle until the sections of the parts being assembled fills the screen. Small, significant motions can then be seen clearly and programmed easily. In contrast, in the real world it is often impractical or impossible to view an assembly process from a preferred angle, or shift one's viewpoint from one side of the assembly to the other because the robot's base is in the way. Finally, safety considerations often prevent the programmer from closeup of real parts held by a robot.

The main advantage of the system's capability for programming intricate motions is the ease of writing the program. It is a simple thing to move the robot images around, teach different intermediate and final points, and have the system generate the text program so that the robots can repeat the motions at will. It is much more difficult to set the actual robots and parts up, and attempt to go through the process of teaching intricate motions using the actual hardware. There is a tendency in the second case to cut corners---to not program-in extra in-

termediate poses, and to settle for a program that already "sort of" works. In the case of off-line programming, it is a simple matter to add intermediate positions, so as to make the insertion proceed well, because the programmer never has to type the instructions. If an insertion works better with, say, fifteen intermediate points, then the programmer trains these fifteen intermediate points. Also, it is significantly easier to reprogram a motion off-line, which means there is no reason to settle for second-best.

c. Quick Reprogramming of Robot-Motion Strategies

In addition to the ability to quickly reprogram a single motion, the system is able to program a completely different type of motion in its place, i.e., changing the strategy of the movement. For instance, the battery insertion was originally accomplished directly: the battery was acquired, positioned directly above the mouth of the case, and inserted straight downwards into place. This worked consistently while the ambient lighting was in its normal position. However, when the lighting was shifted significantly, the system was no longer able to acquire the position of the case accurately, and the straight insertion program failed. To correct for this, a staged insertion was programmed. The battery was tilted 30 degrees or so sideways, forming a tilted surface, in effect a chamfer, with its own side and bottom. The battery was lowered until it contacted the back edge of the case's mouth with its bottom, and then moved forward until it contacted the front edge of the mouth with its side. Next, the battery was rotated back to vertical as it was lowered into the case. In this manner, the robot system was able to deal with more inaccuracy in the location of the case than in the straight insertion.

d. Program Verification

A benefit of off-line programming, which was expected and well realized, is the verification of robot programs by simulation. When a robot program is first written, the programmer expects for the most part that it will execute successfully. However, there are always difficulties involved in working with real robots. To the extent that the simulation models the robots faithfully, the off-line programming system significantly helps in the task of program verification.

The specific areas in which the simulation proves most useful are reachability, singularity determination, and configuration analysis. It is very easy to see, using the simulator, whether a part in a particular pose on the table is reachable by the arm, or whether the wrist must be contorted to approach the part properly. Similarly, it becomes easy to notice, using the simulator, whether a particular movement or grasping operation forces the arm to come close to a joint singularity. If this is the case, then that particular movement or grasping operation can be reprogrammed, the robot can be relocated, or the location of the part can be changed or restricted to another region. It was also useful to verify the robot programs in order to determine the robots' configuration. For example, we started out with the robots in opposite (right and left-handed) configurations. However, after using the simulator to observe the robots work together, it was seen that the robots were getting "tangled up" with each other. The second robot's configuration was changed so that both of the robots worked in a right-handed fashion. This produced much better results.

2. *Difficulties*

The main difficulties encountered with the off-line programming system can be attributed to the lack of power in modeling real-world phenomena. Difficulties include dealing with flexible parts, interpart contact and resulting grasp rotation, weight and slippage, dynamics, robot servoing, and sensing.

a. Flexible Parts

The current off-line programming system has no way of representing flexible parts. This discrepancy was felt mainly with the cover in two particular situations. Although the cover appears to be made out of a relatively rigid plastic, it is mildly deformable. When the robot first grasped the cover and attempted to insert it directly over the top of the flashlight, the insertion failed because the squeeze of the robot's motor-driven fingers, although not extraordinarily firm, distorted the outline of the bottom part of the cover from square to rectangular. This caused the cover to become slightly too narrow to fit the flashlight successfully, and the flashlight was jammed. This problem was corrected by grasping the cover at a different, more reinforced location.

A second problem was encountered with the attempt to model the latches. The latches engage when the cover is lowered to a particular height (i.e., as far down as it can go) above the flashlight and is aligned properly back and forth until it is at the correct orientation. Even then, the latches are "capricious." There was no way to model the random engagement or nonengagement of the latches, and so the off-line programming had to proceed in a feed-forward manner. The robot was programmed to catch one edge of the cover over the light assembly, center and align the cover, and then press downwards. The robot was then lowered to slightly below the minimum height of the cover to provide a small extra force in the insertion. We hoped that this would result in a successful latching, and for the most part, this proved to be the case. So, even though the latching process could not be modeled, operator knowledge served to complete the capabilities of the off-line programming system. and a workable motion program could be generated.

b. Interpart Contact and Grasp Rotation

The WORKMATE system is incapable of modeling the phenomena associated with the contacting of two parts. Interpart contact in general remains a difficult problem. When two parts contact each other, they do not interpenetrate; instead, forces and torques are generated that serve to modify the motion of the contacting parts. In particular, when a part with smooth sides, such as the light assembly, is held in a gripper and pushed against another part, such as the flashlight mouth, the part in the hand will tend to twist and rotate with respect to its former grasp in the fingers.

This cannot be modeled by the existing system. Instead, the system shows the light assembly maintaining a constant grasp relationship with the fingers, and penetrating the model of the flashlight. However, the motion constraint generated by the lens flange enveloping one side of the flashlight mouth and acting like a hinge is a significant part of the insertion. Thus, the operator is forced to move the simulated hand around only, and to visualize what is happening to the constrained light assembly without seeing it depicted in front of him.

An additional and related problem is the fact that there is currently no way to adjust the positions of parts once training a program is under way. The parts stay where they have been placed by the robot. Thus, the battery ends up hovering in the middle of the flashlight, where it was released after its insertion, and the light ends up tilted inside the flashlight case. Although this is a cosmetic detail, it could become significant in a different assembly. An immediate solution is to allow the operator to drag and position parts in the scene, as well as arm joints. A better, but more complex solution, is to model both the effects of gravity and part nonpenetration upon contact, which would result in the system automatically placing the battery in the proper location.

c. Weight and Slippage

Another real-world phenomenon that the system fails to model properly is that of slippage due to weight. During one programming session, the battery was grasped at the top and to the rear of center. A combination of the excessive weight of the battery and its smooth sides rotated the battery around the axis of grasping until its center of gravity was beneath the grasp, and the battery was canted 20 degrees. The system attempted to insert the battery directly into the flashlight, and failed, because the simulation did not model the actual locations of the parts faithfully.

d. Dynamics

An unanticipated problem stemmed from the fact that parts slip differently when the robot moves at different speeds. In particular, parts that do not slip at all when the robot is moving at slow speeds can suddenly change to flying out of the hand when the robot is moving at faster speeds. The battery was particularly pernicious in this manner, due to its smooth sides and its relatively large mass. The current simulation is based on kinematics, and does not take dynamics into consideration. In the end, we simply had to restrict the execution speeds of the robots to values slow enough to ensure successful transfers. This problem may be overcome by modeling the dynamics and friction associated with the objects.

e. Robot Servoing

There are two problems with servoing the real robot: moving too close to joint singularities and attempting to change configurations. These problems are caused by changes in the way the program is executed (although the programs themselves do not change). The changes are a result of parts being relocated, and the fact that the system can adjust its execution to go along with these relocations by using part-relative motions and sensory feedback to determine the part locations, inside the program itself, as travel off-line. For instance, in a straight-line motion near a singularity, the controller may command the robot to move too quickly in an imprecise manner. In addition, especially if the verifier is not used to extensively check programs out under enough different circumstances, it is pos-

sible to train programs on the simulator that will not execute on the real robot controller because the current position of the parts would cause the robot to change configurations. A temporary solution is to ensure that the parts are in some initial configuration that does not produce these problems. A long-term solution is to create an intelligent and flexible execution interpreter that can foresee such difficulties and modify the arm paths in real time to circumvent them.

f. Sensing

The biggest problems we had were with sensing. Since the current system does not have a vision-module simulator, it is unable to predict accurately where the center of gravity of the two-dimensional silhouette of a part, returned by the vision module, is located. Instead, the simulation system uses the center of the modeled part as its definition, which requires that a calibrated adjustment be made on the value returned by the actual vision system. This is undesirable because it slows down the operator and is difficult to determine; furthermore, it can introduce inaccuracies into the system. A vision-module simulator would automate this process. In addition to this problem, the usual sensor difficulties such as transparent parts, stray reflections, and parallax distortions, caused problems in the execution of the robot programs. Figure 10 shows the actual vision module's view of the parts on the light table. Notice that the transparent rim of the light assembly in lower right has completely disappeared. A facility to model the vision module's processing of images would at least eliminate or call attention to the known problems.

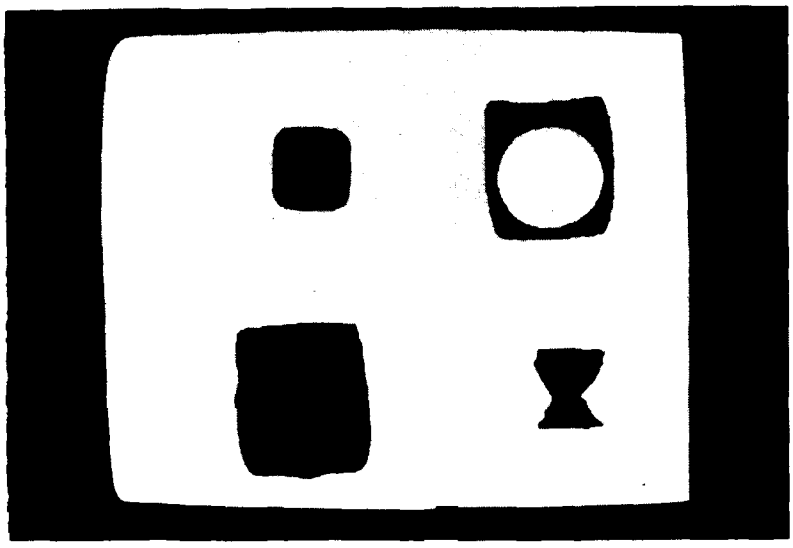


Figure 10: The Vision Module's View of the Four Parts

E. Conclusions

In this project, we explored the problems of building and working with a robotic off-line programming system. We demonstrated the actual use of the system with a robotic testbed: not only were we able to write robot programs, but we were able to successfully execute them as well. The robot programs included the use of sensory feedback to determine part locations that were unknown before on-line execution. Many successes and problem areas were identified. An important point is that the execution of the robot programs does not work well when the simulation fails to model the physical phenomena of the actual system.

There is much left to be done in the area of exploring a general off-line programming system. A mandatory package that needs to be added is an editor that can model parts and move them around and display them under user control. In addition, there is still much work to be done in the program editor; the abilities to

reedit programs, to randomly edit movements, to branch and loop, and to call robot subroutines, are all lacking. The lacks in sensor simulation have already been discussed. Finally, the difficult problem of contact simulation, and the resulting closed-kinematic-chain motion problems required for a full contact simulation, remain as challenges to be worked on.

Appendix A

PUBLICATIONS

Appendix A

PUBLICATIONS

(1) "On the Representation and Estimation of Spatial Uncertainty," by R.C. Smith and P. Cheeseman; accepted for publication by the *International Journal of Robotics Research*.

(2) "Estimating Uncertain Spatial Relationships in Robotics," by R.C. Smith, M. Self, and P. Cheeseman; accepted for publication in the proceedings of the workshop, *Uncertainty in Artificial Intelligence*, to be held in Philadelphia, Pennsylvania, August 1986.

(3) "Fast Robot Collision Detection Using Graphics Hardware," by R.C. Smith; published in the *Proceedings of the Symposium on Robot Control (SYROCO)*, Barcelona, Spain, 1985.

(4) "Determining An Object's Location in a Robot's Hand By Means of Vision," by Eitan Zeiler, Robotics Laboratory Technical Note, SRI International, August 1984.

(5) "Estimating Object Location In A Manipulator's Hand Using Force/Torque Information," by A. Bergman and R.C. Smith, to be submitted for publication.

(6) "Robot WORKMATE: Interactive-Graphic Off-Line Programming," by R.C. Smith, Robotics Laboratory Technical Note, SRI International, February 1986.

Appendix B

PERSONNEL

Appendix B

PERSONNEL

Principal Investigators:

Randall C. Smith, Research Engineer, Robotics Laboratory
David Nitzan, Director, Robotics Laboratory

Basic Problems:

Locational Uncertainty
Randall C. Smith, Research Engineer
Peter Cheeseman, Senior Computer Scientist

Spatial Reasoning
Randall C. Smith, Research Engineer

Sensor Usage
Eitan Zeiler, International Fellow
Aviv Bergman, Research Physicist

WORKMATE System Development

Modeling
Randall C. Smith, Research Engineer

On-Line Programming
Randall C. Smith, Research Engineer

Off-Line Programming
Randall C. Smith, Research Engineer
John K. Myers, Research Engineer

Experimental Verification
Randall C. Smith, Research Engineer
John K. Myers, Research Engineer
Antony J. Sword, Research Engineer

Appendix C

DISTRIBUTION LIST

Appendix C

DISTRIBUTION LIST

AFOSR

Lt. Col Harry V. Winsor
Acting Director
Electronic and Material Sciences
Air Force Office of Scientific Research
Bolling Air Force Base, DC 20332

Air Force

Ted Brandewie
Air Force Wright Aeronautical Laboratories
Materials Laboratory
Manufacturing Technology Division
Wright-Patterson Air Force Base, Ohio 45433

Lt. Michael F. Hitchcock
Project Leader
Manufacturing Sciences Program
Air Force Materials Laboratory (AFWAL/ML)
Wright-Patterson Air Force Base, Ohio 45433

Dr. Vincent J. Russo
Director
Manufacturing Science Program
Air Force Materials Laboratory (AFWAL/ML)
Wright-Patterson Air Force Base, Ohio 45433

DARPA

Dr. William E. Isler
Program Manager, Robotic Systems
Systems Sciences Division
Defense Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Clinton Kelly
Director, Defense Sciences Office
Defense Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

Robert Rosenfeld
Defense Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

Army

Mr. Timothy Evans
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27701

Navy

LCDR Bart Everett
Special Assistant for Robotics
Naval Sea Systems Command, C90-M
Washington, DC 20362

Alan Meyrowitz
Office of Naval Research
Code 437
800 N. Quincy
Arlington, VA 22217

NSF

Mr. Norman Caplan
Program Manager, Elec., Comp., and System Eng.
National Science Foundation
1800 G. Street, N.W.
Washington, DC 20550

Howard Moraff
Program Director
Automation and Systems Interpertation
1800 G. Street, N.W.
Washington, DC 20550

Bernard Chern
National Science Foundation
1800 G. Street, N.W.
Washington, DC 20550

NBS

Dr. James Albus
Division Chief, Industrial Systems Division
Center for Manufacturing Engineering
National Bureau of Standards
Bldg. 220, Room A123
Washington, DC 20234

Stanford University

Dr. Thomas O. Binford
Department of Computer Science
Stanford University
Stanford, CA 94305

Dr. Robert H. Cannon
Stanford University
250 Durand Bldg.
Stanford, CA 94305

University of Michigan

Dean Daniel Atkins
University of Michigan
Chrysler Center
Ann Arbor, Michigan 48109

Brigham Young University

Del Allen
Brigham Young University
Provo, Utah 84602

Appendix D

ESTIMATING UNCERTAIN SPATIAL RELATIONSHIPS

Estimating Uncertain Spatial Relationships in Robotics

Randall Smith*

Matthew Self†

Peter Cheeseman†

SRI International
333 Ravenswood Avenue
Menlo Park, California 94025

The research reported in this paper was supported by the National Science Foundation under Grant ECS-8200615, the Air Force Office of Scientific Research under Contract F49620-84-K-0007, and by General Motors Research Labs.

Abstract

In many robotic applications the need to represent and reason about spatial relationships is of great importance. However, our knowledge of particular spatial relationships is inherently uncertain. The most used method for handling the uncertainty is to "pre-engineer" the problem away, by structuring the working environment and using specially-suited high-precision equipment. In some advanced robotic research domains, however, such as automatic task planning, off-line robot programming, and autonomous vehicle operation, prior structuring will not be possible, because of dynamically changing environments, or because of the demand for greater reasoning flexibility. Spatial reasoning is further complicated because relationships are often not described explicitly, but are given by uncertain *relative* information. This is particularly true when many different frames of reference are used, producing a network of uncertain relationships. Rather than treat spatial uncertainty as a side issue in geometrical reasoning, we believe it must be an intrinsic part of spatial representations. In this paper, we describe a representation for spatial information, called the *stochastic map*, and associated procedures for building it, reading information from it, and revising it incrementally as new information is obtained. The map always contains the best estimates of relationships among objects in the map, and their uncertainties. The procedures provide a general solution to the problem of estimating uncertain relative spatial relationships. The estimates are probabilistic in nature, an advance over the previous, very conservative, worst-case approaches to the problem. Finally, the procedures are developed in the context of state-estimation and filtering theory, which provides a solid basis for numerous extensions.

1 Introduction

In many applications of robotics, such as industrial automation, and autonomous vehicles, there is a need to represent and reason about spatial uncertainty. In the past, this need has been circumvented by special purpose methods such as precision engineering, very accurate sensors and the use of fixtures and calibration points. While these methods sometimes supply sufficient accuracy to avoid the need to represent uncertainty explicitly, they are usually costly. An alternative approach is to use multiple, overlapping, lower resolution sensors and to combine the spatial information (including the uncertainty) from all sources to obtain the best spatial estimate. This integrated information can often supply sufficient accuracy to avoid the need for the hard engineered approach.

In addition to lower hardware cost, the explicit

*Currently at General Motors Research Labs, Warren, Michigan.

†Currently at NASA Ames Research Ctr., Moffett Field, California.

Accepted for publication in the proceedings of the workshop *Uncertainty in Artificial Intelligence*, Philadelphia, Pennsylvania, August 1986. To be submitted for publication in the *International Journal of Robotics Research*.

estimation of uncertain spatial information makes it possible to decide *in advance* whether proposed operations are likely to fail because of accumulated uncertainty, and whether proposed sensor information will be sufficient to reduce the uncertainty to tolerable limits. In other situations, such as inexpensive mobile robots, the *only* way to obtain sufficient accuracy is to combine the (uncertain) information from many sensors.

A difficulty in combining uncertain spatial information is that it often occurs in the form of uncertain *relative* information. This is particularly true where many different frames of reference are used, and the uncertain spatial information must be propagated between these frames. This paper presents a general solution to the problem of estimating uncertain spatial relationships, regardless of which frame the information is presented in, or in which frame the answer is required. The basic theory assumes that the errors are "small", so that the nonlinear transformations from one frame to another are approximately linear.

Early methods for representing spatial uncertainty (e.g. [Taylor, 1976]) numerically computed min-max bounds on errors in typical robotics applications. Brooks extended this analysis to symbolically computing min-max bounds [Brooks, 1982]. This min-max approach is very conservative compared to the probabilistic approach in this paper, because it always assumes the worst case when combining information. More recently, a probabilistic representation of uncertainty was developed for the HILARE robot [Chatila, 1985] that is similar to the method presented here, except that it uses only a scalar representation of positional uncertainty instead of a multivariate one. In a recent paper, Brooks developed a representation of spatial uncertainty based on bounding cylinders and a combining operation based on the intersections of such cylinders [Brooks, 1985]. Smith and Cheeseman ([Smith, 1984], [Smith, 1985]), working on problems in off-line programming of industrial automation tasks, proposed operations that could reduce graphs of uncertain relationships (represented by multivariate probability distributions) to a single, best estimate of some relationship of interest. The current paper extends that work, but in the formal setting of estimation theory, and does not utilise graph transformations.

In summary, many important applications require a representation of spatial uncertainty. In addition, methods for combining uncertain spatial information and transforming such information from one frame to another are required. This paper presents a matrix representation of spatial uncertainty that explicitly represents the uncertainty for each degree of freedom in the world of interest. A method is given for combining uncertain information regardless of which frame it is presented in, and it allows the description of the spatial uncertainty of one frame relative to any other frame. The necessary procedures are presented in matrix form, suitable for efficient implementation. In particular, methods are given for incrementally building the best estimate "map" and its uncertainty as new pieces of uncertain spatial information are added.

2 The Stochastic Map

Our knowledge of the spatial relationships among objects is inherently uncertain. A man-made object does not match its geometric model *exactly* because of manufacturing tolerances. *Even if it did*, a sensor could not measure the geometric features, and thus locate the object *exactly*, because of measurement errors. And *even if it could*, a robot using the sensor cannot manipulate the object *exactly* as intended, because of hand positioning errors. These errors can be reduced to neglible limits for some tasks, by "pre-enginerring" the solution — structuring the working environment and using specially-suited high-precision equipment — but at great cost of time and expense.

However, rather than treat spatial uncertainty as a side issue in geometrical reasoning, we believe it must be treated as an intrinsic part of spatial representations.

In this paper, uncertain spatial relationships will be tied together in a representation called the *stochastic map*. It contains estimates of the spatial relationships, their uncertainties, and their interdependencies.

First, the map structure will be described, followed by methods for extracting information from

it. Finally, a procedure will be given for building the map *incrementally*, as new spatial information is obtained.

To illustrate the theory, we will present an example of a mobile robot acquiring knowledge about its location and the organisation of its environment by making sensor observations at different times and in different places.

2.1 Representation

In order to formalise the above ideas, we will define the following terms. A *spatial relationship* will be represented by the vector of its *spatial variables*, $\mathbf{x}$. For example, the position and orientation of a mobile robot can be described by its coordinates, x and y, in a two dimensional cartesian reference frame and by its orientation, ϕ, given as a rotation about the z axis:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \phi \end{bmatrix}.$$

An *uncertain* spatial relationship, moreover, can be represented by a *probability distribution* over its spatial variables — i.e., by a probability density function that assigns a probability to each particular combination of the spatial variables, $\mathbf{x}$:

$$P(\mathbf{x}) = f(\mathbf{x})d\mathbf{x}.$$

Such detailed knowledge of the probability distribution is usually unneccesary for making decisions, such as whether the robot will be able to complete a given task (e.g. passing through a doorway). Furthermore, most measuring devices provide only a nominal value of the measured relationship, and we can estimate the average error from the sensor specifications. For these reasons, we choose to model an uncertain spatial relationship by estimating the first two moments of its probability distribution— the *mean*, $\hat{\mathbf{x}}$ and the *covariance*, $\mathbf{C}(\mathbf{x})$, defined as:

$$\begin{aligned} \hat{\mathbf{x}} &\triangleq E(\mathbf{x}), \\ \tilde{\mathbf{x}} &\triangleq \mathbf{x} - \hat{\mathbf{x}}, \\ \mathbf{C}(\mathbf{x}) &\triangleq E(\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T). \end{aligned} \tag{1}$$

where E is the expectation operator, and $\tilde{\mathbf{x}}$ is the deviation from the mean.

For our mobile robot example, these are:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{\phi} \end{bmatrix}, \qquad \mathbf{C}(\mathbf{x}) = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\phi} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{y\phi} \\ \sigma_{x\phi} & \sigma_{y\phi} & \sigma_\phi^2 \end{bmatrix}.$$

Here, the diagonal elements of the covariance matrix are just the variances of the spatial variables, while the off-diagonal elements are the covariances between the spatial variables. It is useful to think of the covariances in terms of their correlation coefficients, ρ_{ij}:

$$\rho_{ij} \triangleq \frac{\sigma_{ij}}{\sigma_i \sigma_j} = \frac{E(\tilde{x}_i \tilde{x}_j)}{\sqrt{E(\tilde{x}_i^2)E(\tilde{x}_j^2)}}, \qquad -1 \le \rho_{ij} \le 1.$$

Similarly, to model a system of n uncertain spatial relationships, we construct the vector of *all* the spatial variables, which we call the *system state vector*. As before, we will estimate the mean of the state vector, $\hat{\mathbf{x}}$, and the *system covariance matrix*, $\mathbf{C}(\mathbf{x})$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \quad \hat{\mathbf{x}} = \begin{bmatrix} \hat{\mathbf{x}}_1 \\ \hat{\mathbf{x}}_2 \\ \vdots \\ \hat{\mathbf{x}}_n \end{bmatrix}, \quad \mathbf{C}(\mathbf{x}) =$$

$$\begin{bmatrix} \mathbf{C}(\mathbf{x}_1) & \mathbf{C}(\mathbf{x}_1,\mathbf{x}_2) & \cdots & \mathbf{C}(\mathbf{x}_1,\mathbf{x}_n) \\ \mathbf{C}(\mathbf{x}_2,\mathbf{x}_1) & \mathbf{C}(\mathbf{x}_2) & \cdots & \mathbf{C}(\mathbf{x}_2,\mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}(\mathbf{x}_n,\mathbf{x}_1) & \mathbf{C}(\mathbf{x}_n,\mathbf{x}_2) & \cdots & \mathbf{C}(\mathbf{x}_n) \end{bmatrix} \tag{2}$$

where:

$$\begin{aligned} \mathbf{C}(\mathbf{x}_i,\mathbf{x}_j) &\triangleq E(\tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_j^T), \\ \mathbf{C}(\mathbf{x}_j,\mathbf{x}_i) &= \mathbf{C}(\mathbf{x}_i,\mathbf{x}_j)^T. \end{aligned} \tag{3}$$

Here, the $\mathbf{x}_i$'s are the vectors of the spatial variables of the individual uncertain spatial relationships, and the $\mathbf{C}(\mathbf{x}_i)$'s are the associated covariance matrices, as discussed earlier. The $\mathbf{C}(\mathbf{x}_i,\mathbf{x}_j)$'s

are the cross-covariance matrices between the uncertain spatial relationships, which allow for dependencies between the uncertainties of different spatial relationships. These off-diagonal matrices provide the mechanism for back-propagating new information added to the map, in order to improve previous spatial estimates, and are significantly more sophisticated than previous methods for doing this.

In our example, each uncertain spatial relationship is of the same form, so $\mathbf{x}$ has $m = 3n$ elements, and we may write:

$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \\ \phi_i \end{bmatrix}, \qquad \hat{\mathbf{x}}_i = \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \\ \hat{\phi}_i \end{bmatrix},$$

$$\mathbf{C}(\mathbf{x}_i, \mathbf{x}_j) = \begin{bmatrix} \sigma_{x_i x_j} & \sigma_{x_i y_j} & \sigma_{x_i \phi_j} \\ \sigma_{x_i y_j} & \sigma_{y_i y_j} & \sigma_{y_i \phi_j} \\ \sigma_{x_i \phi_j} & \sigma_{y_i \phi_j} & \sigma_{\phi_i \phi_j} \end{bmatrix}.$$

Thus our "map" consists of the current estimate of the mean of the system state vector, which gives the nominal locations of objects in the map with respect to the world reference frame, and the associated system covariance matrix, which gives the uncertainty of each point in the map and the interdependencies of these uncertainties.

2.2 Interpretation

For some decisions based on uncertain spatial relationships, we must assume a particular distribution that fits the estimated moments. For example, a robot might need to be able to calculate the probability that a ceratin object will be in its field of view, or the probability that it will succeed in passing through a doorway.

Given only the mean, $\mathbf{x}$, and covariance matrix, $\mathbf{C}(\mathbf{x})$, of a multivariate probability distribution, the principle of maximum entropy indicates that the distribution which assumes the least information is the normal distribution. Furthermore if the spatial relationship is calculated by combining evidence from many independent observations, the central limit theorem indicates that the resulting distribution will tend to a normal distribution:

$$P(\mathbf{x}) = \frac{\exp\left[-\frac{1}{2}(\mathbf{x}-\hat{\mathbf{x}})^T \mathbf{C}^{-1}(\mathbf{x})(\mathbf{x}-\hat{\mathbf{x}})\right]}{\sqrt{(2\pi)^m |\mathbf{C}(\mathbf{x})|}} d\mathbf{x}. \quad (4)$$

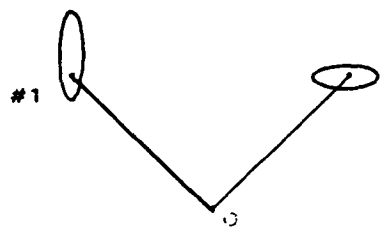
We will graph uncertain spatial relationships by plotting contours of constant probability from a normal distribution with the given mean and covariance information. These contours turn out to be concentric ellipsoids (ellipses for two dimensions) whose parameters can be calculated from the covariance matrix, $\mathbf{C}(\mathbf{x}_i)$ [Nahi, 1976]. It is important to emphasize that we do not assume that the uncertain spatial relationships are described by normal distributions. We estimate the mean and variance of their distributions, and use the normal distribution only when we need to calculate specific probability contours.

In the figures in this paper, the plotted points show the *actual* locations of objects, which are known only by the simulator and displayed for our benefit. The robot's information is shown by the ellipses which are drawn centered on the estimated mean of the relationship and such that they enclose a 99.9% confidence region (about four standard deviations) for the relationships.

2.3 Example

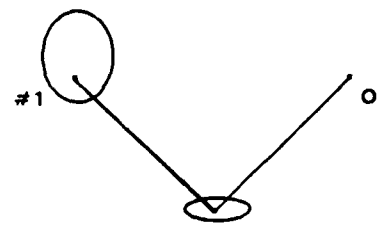
Throughout this paper we will refer to a two dimensional example involving the navigation of a mobile robot with three degrees of freedom. In this example the robot performs the following sequence of actions:

- The robot senses object #1
- The robot moves.
- The robot senses an object (object #2) which it determines cannot be object #1.
- Trying again, the robot succeeds in sensing object #1, thus helping to localize itself, object #1, and object #2.



THE ROBOT SENSES OBJECT #1 AND MOVES

Figure 1:



THE WORLD FROM THE ROBOT'S NEW FRAME

Figure 2:

Figure 1 shows two examples of uncertain spatial relationships — the sensed location of object #1, and the end-point of a planned motion for the robot. The robot is initially sitting at the location marked 'O'. There is enough information in our stochastic map at this point for the robot to be able to decide how likely a collision with the object is, if the motion is made. In this case the probability is vanishingly small.

Figure 2 shows how this spatial knowledge can be presented from the robot's new reference frame after its motion. As expected, the uncertainty in the location of object #1 becomes larger when it is compounded with the uncertainty in the robot's motion.

From this new location, the robot senses object #2 (Figure 3). The robot is able to determine with the information in its stochastic map that this must be a new object and is not object #1 which it observed earlier.

In figure 4, the robot senses object #1 again. This new, *loop closing* sensor measurement acts as a constraint, and is incorporated into the map, reducing the uncertainty in the locations of the robot, object #1 *and* Object #2 (Figure 5).

3 Reading the Map

3.1 Uncertain Relationships

Having seen how we can represent uncertain spatial relationships by estimates of the mean and covariance of the system state vector, we now discuss methods for estimating the first two moments of unknown multivariate probability distributions. See [Papoulis, 1965] for detailed justifications of the following topics.

3.1.1 Linear Relationships

The simplest case concerns relationships which are linear in the random varables, e.g.:

$$\mathbf{y} = \mathbf{M}\mathbf{x} + \mathbf{b},$$

where, $\mathbf{x}$ $(n \times 1)$ is a random vector, $\mathbf{M}$ $(r \times n)$ is the *non*-random coefficient matrix, $\mathbf{b}$ $(r \times 1)$ is a constant vector, and $\mathbf{y}$ $(r \times 1)$ is the resultant random vector. Using the definitions from (1), and the linearity of the expectation operator, E, one can easily verify that the mean of the relationship, $\hat{\mathbf{y}}$, is given by:

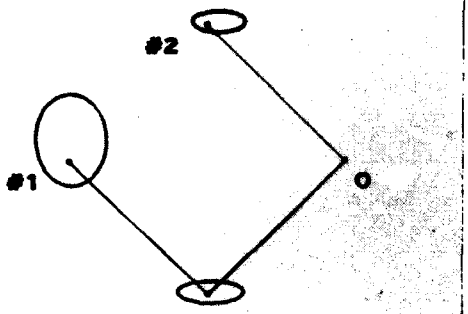
$$\hat{\mathbf{y}} = \mathbf{M}\hat{\mathbf{x}} + \mathbf{b}, \tag{5}$$

and the covariance matrix, $\mathbf{C}(\mathbf{y})$, is:

$$\mathbf{C}(\mathbf{y}) = \mathbf{M}\mathbf{C}(\mathbf{x})\mathbf{M}^T. \tag{6}$$

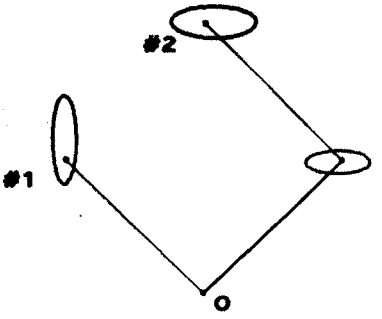
We will also need to be able to compute the covariance between $\mathbf{y}$ and some other relationship, $\mathbf{z}$, given the covariance between $\mathbf{x}$ and $\mathbf{z}$:

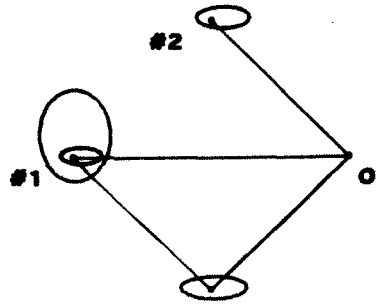
$$\begin{aligned} \mathbf{C}(\mathbf{y},\mathbf{z}) &= \mathbf{M}\mathbf{C}(\mathbf{x},\mathbf{z}), \\ \mathbf{C}(\mathbf{z},\mathbf{y}) &= \mathbf{C}(\mathbf{z},\mathbf{x})\mathbf{M}^T. \end{aligned} \tag{7}$$



THE ROBOT SENSES OBJECT #2

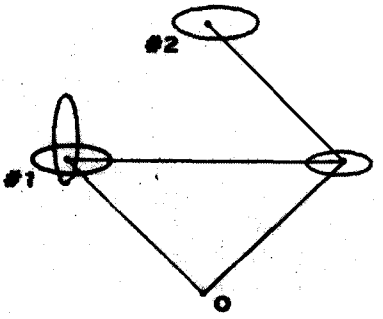
Figure 3: OBJECT #2 FROM THE WORLD FRAME

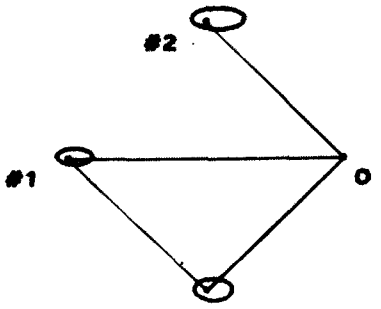




THE ROBOT SENSES OBJECT #1 AGAIN

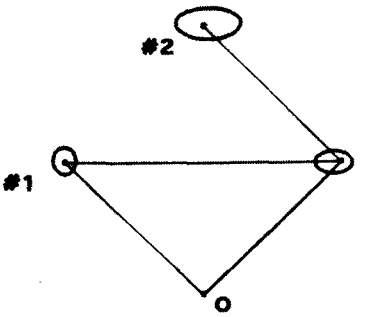
Figure 4: OBJECT #1 FROM THE WORLD FRAME





THE UPDATE FROM THE ROBOT'S FRAME

Figure 5: THE UPDATE FROM THE WORLD FRAME



The first two moments of the multivariate distribution of $\mathbf{y}$ are computed exactly, given correct moments for $\mathbf{x}$. Further, if $\mathbf{x}$ follows a normal distribution, then so does $\mathbf{y}$.

3.1.2 Non-Linear Relationships

The first two moments computed by the formulae below for non-linear relationships on random variables will be first-order estimates of the true values. To compute the actual values requires knowledge of the *complete* probability density function of the spatial variables, which will not generally be available in our applications. The usual approach is to approximate the non-linear function

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

by a Taylor series expansion about the estimated mean, $\hat{\mathbf{x}}$, yielding:

$$\mathbf{y} = \mathbf{f}(\hat{\mathbf{x}}) + \mathbf{F}_{\mathbf{x}}\tilde{\mathbf{x}} + \cdots,$$

where $\mathbf{F}_{\mathbf{x}}$ is the matrix of partials, or Jacobian, of $\mathbf{f}$ evaluated at $\hat{\mathbf{x}}$:

$$\mathbf{F}_{\mathbf{x}} \triangleq \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}(\hat{\mathbf{x}}) \triangleq \left[\begin{array}{cccc} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_r}{\partial x_1} & \frac{\partial f_r}{\partial x_2} & \cdots & \frac{\partial f_r}{\partial x_n} \end{array}\right]_{\mathbf{x}=\hat{\mathbf{x}}}$$

This terminology is the extension of the f_x terminology from scalar calculus to vectors. The Jacobians are always understood to be evaluated at the estimated mean of the input variables.

Truncating the expansion for $\mathbf{y}$ after the linear term, and taking the expectation produces the linear estimate of the mean of $\mathbf{y}$:

$$\hat{\mathbf{y}} \approx \mathbf{f}(\hat{\mathbf{x}}). \tag{8}$$

Similarly, the first-order estimate of the covariances are:

$$\begin{aligned} \mathbf{C}(\mathbf{y}) &\approx \mathbf{F}_{\mathbf{x}}\mathbf{C}(\mathbf{x})\mathbf{F}_{\mathbf{x}}^T, \\ \mathbf{C}(\mathbf{y},\mathbf{z}) &\approx \mathbf{F}_{\mathbf{x}}\mathbf{C}(\mathbf{x},\mathbf{z}), \\ \mathbf{C}(\mathbf{z},\mathbf{y}) &\approx \mathbf{C}(\mathbf{z},\mathbf{x})\mathbf{F}_{\mathbf{x}}^T. \end{aligned} \tag{9}$$

Though not utilized in our application, the second order term may be included in the Taylor series expansion to improve the mean estimate:

$$\mathbf{y} = \mathbf{f}(\hat{\mathbf{x}}) + \mathbf{F}_{\mathbf{x}}\tilde{\mathbf{x}} + \frac{1}{2}\mathbf{F}_{\mathbf{xx}}(\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T) + \cdots,$$

We denote the (3 dimensional) matrix of second partials of $\mathbf{f}$ by $\mathbf{F}_{\mathbf{xx}}$. To avoid uneccesary complexity, we simply state that the ith element of the vector produced when $\mathbf{F}_{\mathbf{xx}}$ is multiplied on the right by a matrix $\mathbf{A}$ is defined by:

$$(\mathbf{F}_{\mathbf{xx}}\mathbf{A})_i = trace\left[\left(\frac{\partial^2 f_i}{\partial x_j \partial x_k}\bigg|_{\mathbf{x}=\hat{\mathbf{x}}}\right)\mathbf{A}\right].$$

The second-order estimate of the mean of $\mathbf{y}$ is then:

$$\hat{\mathbf{y}} \approx \mathbf{f}(\hat{\mathbf{x}}) + \frac{1}{2}\mathbf{F}_{\mathbf{xx}}\mathbf{C}(\mathbf{x}),$$

and the second-order estimate of the covariance is:

$$\mathbf{C}(\mathbf{y}) \approx \mathbf{F}_{\mathbf{x}}\mathbf{C}(\mathbf{x})\mathbf{F}_{\mathbf{x}}^T - \frac{1}{4}\mathbf{F}_{\mathbf{xx}}\mathbf{C}(\mathbf{x})\mathbf{C}(\mathbf{x})^T\mathbf{F}_{\mathbf{xx}}^T.$$

In the remainder of this paper we consider only first order estimates, and the symbol "$\approx$" should read as "linear estimate of."

3.2 Spatial Relationships

We now consider the actual spatial relationships which are most often encountered in robotics applications. We will develop our presentation about the three degree of freedom formulae, since they suit our examples concerning a mobile robot. Formulae for the three dimensional case with six degrees of freedom are given in Appendix A.

3.2.1 Compounding

Given two spatial relationships, $\mathbf{x}_{ij}$ and $\mathbf{x}_{jk}$, as in Figure 2, we wish to compute the resultant relationship $\mathbf{x}_{ik}$. The formula for computing $\mathbf{x}_{ik}$ from $\mathbf{x}_{ij}$ and $\mathbf{x}_{jk}$ is:

$$\begin{aligned} \mathbf{x}_{ik} &\triangleq \mathbf{x}_{ij} \oplus \mathbf{x}_{jk} \\ &= \begin{bmatrix} x_{jk}\cos\phi_{ij} - y_{jk}\sin\phi_{ij} + x_{ij} \\ x_{jk}\sin\phi_{ij} + y_{jk}\cos\phi_{ij} + y_{ij} \\ \phi_{ij} + \phi_{jk} \end{bmatrix}. \end{aligned}$$

We call this operation compounding, and it is used to calculate the resultant relationship from two given relationships which are arranged head-to-tail. It would be used, for instance, to determine the location of a mobile robot after a sequence of relative motions. Remember that these transformations involve rotations, so compounding is not merely vector addition.

Utilising (8), the first-order estimate of the mean of the compounding operation is:

$$\hat{\mathbf{x}}_{ik} \approx \hat{\mathbf{x}}_{ij} \oplus \hat{\mathbf{x}}_{jk}.$$

Also, from (9), the first-order estimate of the covariance is:

$$\mathbf{C}(\mathbf{x}_{ik}) \approx \mathbf{J}_{\oplus} \begin{bmatrix} \mathbf{C}(\mathbf{x}_{ij}) & \mathbf{C}(\mathbf{x}_{ij}, \mathbf{x}_{jk}) \\ \mathbf{C}(\mathbf{x}_{jk}, \mathbf{x}_{ij}) & \mathbf{C}(\mathbf{x}_{jk}) \end{bmatrix} \mathbf{J}_{\oplus}^T.$$

where the Jacobian of the compounding operation, $\mathbf{J}_{\oplus}$ is given by:

$$\mathbf{J}_{\oplus} \triangleq \frac{\partial(\mathbf{x}_{ij} \oplus \mathbf{x}_{jk})}{\partial(\mathbf{x}_{ij}, \mathbf{x}_{jk})} = \frac{\partial \mathbf{x}_{ik}}{\partial(\mathbf{x}_{ij}, \mathbf{x}_{jk})} =$$

$$\begin{bmatrix} 1 & 0 & -(y_{ik} - y_{ij}) & \cos\phi_{ij} & -\sin\phi_{ij} & 0 \\ 0 & 1 & (x_{ik} - x_{ij}) & \sin\phi_{ij} & \cos\phi_{ij} & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Note how we have utilised the resultant relationship $\mathbf{x}_{ik}$ in expressing the Jacobian. This results in greater computational efficiency than expressing the Jacobian only in terms of the compounded relationships $\mathbf{x}_{ij}$ and $\mathbf{x}_{jk}$. We can always estimate the mean of an uncertain relationship and then use this result when evaluating the Jacobian to estimate the covariance of the relationship.

In the case that the two relationships being compounded are independent ($\mathbf{C}(\mathbf{x}_{ij}, \mathbf{x}_{jk}) = 0$), we can rewrite the first-order estimate of the covariance as:

$$\mathbf{C}(\mathbf{x}_{ik}) \approx \mathbf{J}_{1\oplus}\mathbf{C}(\mathbf{x}_{ij})\mathbf{J}_{1\oplus}^T + \mathbf{J}_{2\oplus}\mathbf{C}(\mathbf{x}_{jk})\mathbf{J}_{2\oplus}^T$$

where $\mathbf{J}_{1\oplus}$ and $\mathbf{J}_{2\oplus}$ are the left and right halves (3×3) of the compounding Jacobian (3×6):

$$\mathbf{J}_{\oplus} = \begin{bmatrix} \mathbf{J}_{1\oplus} & \mathbf{J}_{2\oplus} \end{bmatrix}.$$

3.2.2 The Inverse Relationship

Given a relationship $\mathbf{x}_{ij}$, the formula for the coordinates of the inverse relationship $\mathbf{x}_{ji}$, as a function of $\mathbf{x}_{ij}$ is:

$$\mathbf{x}_{ji} \triangleq \ominus\mathbf{x}_{ij} \triangleq \begin{bmatrix} -x_{ij}\cos\phi_{ij} - y_{ij}\sin\phi_{ij} \\ x_{ij}\sin\phi_{ij} - y_{ij}\cos\phi_{ij} \\ -\phi_{ij} \end{bmatrix}.$$

We call this the reverse relationship. Using (8) we get the first-order mean estimate:

$$\hat{\mathbf{x}}_{ji} \approx \ominus\hat{\mathbf{x}}_{ij}.$$

and from (9) the first-order covariance estimate is:

$$\mathbf{C}(\mathbf{x}_{ji}) \approx \mathbf{J}_{\ominus}\mathbf{C}(\mathbf{x}_{ij})\mathbf{J}_{\ominus}^T.$$

where the Jacobian for the reversal operation, $\mathbf{J}_{\ominus}$ is:

$$\mathbf{J}_{\ominus} \triangleq \frac{\partial \mathbf{x}_{ji}}{\partial \mathbf{x}_{ij}} = \begin{bmatrix} -\cos\phi_{ij} & -\sin\phi_{ij} & y_{ji} \\ \sin\phi_{ij} & -\cos\phi_{ij} & -x_{ji} \\ 0 & 0 & -1 \end{bmatrix}.$$

Note that the uncertainty is not inverted, but rather expressed from the opposite (reverse) point of view.

3.2.3 Composite Relationships

We have shown how to compute the resultant of two relationships which are arranged head-to-tail, and also how to reverse a relationship. With these two operations we can calculate the resultant of any sequence of relationships.

For example, the resultant of a chain of relationships arranged head-to-tail can be computed recursively by:

$$\begin{aligned} \mathbf{x}_{il} &= \mathbf{x}_{ij} \oplus \mathbf{x}_{jl} = \mathbf{x}_{ij} \oplus (\mathbf{x}_{jk} \oplus \mathbf{x}_{kl}) \\ &= \mathbf{x}_{ik} \oplus \mathbf{x}_{kl} = (\mathbf{x}_{ij} \oplus \mathbf{x}_{jk}) \oplus \mathbf{x}_{kl} \end{aligned}$$

Note, the compounding operation is associative, but not commutative.

We have denoted the reversal operation by $\ominus$ so that by analogy to conventional + and − we may write:

$$\mathbf{x}_{ij} \ominus \mathbf{x}_{kj} \triangleq \mathbf{x}_{ij} \oplus (\ominus \mathbf{x}_{kj}) .$$

This is the head-to-head combination of two relationships.

The tail-to-tail combination arises quite often (as in figure 1), and is given by:

$$\mathbf{x}_{jk} = \ominus \mathbf{x}_{ij} \oplus \mathbf{x}_{ik}$$

To estimate the mean of a complex relationship, such as the tail-to-tail combination, we merely solve the estimate equations recursively:

$$\hat{\mathbf{x}}_{jk} = \hat{\mathbf{x}}_{ji} \oplus \hat{\mathbf{x}}_{ik} = \ominus \hat{\mathbf{x}}_{ij} \oplus \hat{\mathbf{x}}_{ik}$$

The covariance can be estimated in a similar way:

$$\mathbf{C}(\mathbf{x}_{jk}) \approx \mathbf{J}_{\oplus} \begin{bmatrix} \mathbf{C}(\mathbf{x}_{ji}) & \mathbf{C}(\mathbf{x}_{ji}, \mathbf{x}_{ik}) \\ \mathbf{C}(\mathbf{x}_{ik}, \mathbf{x}_{ji}) & \mathbf{C}(\mathbf{x}_{ik}) \end{bmatrix} \mathbf{J}_{\oplus}^T$$

$$\approx \mathbf{J}_{\oplus} \begin{bmatrix} \mathbf{J}_{\ominus} \mathbf{C}(\mathbf{x}_{ij}) \mathbf{J}_{\ominus}^T & \mathbf{C}(\mathbf{x}_{ij}, \mathbf{x}_{ik}) \mathbf{J}_{\ominus}^T \\ \mathbf{J}_{\ominus} \mathbf{C}(\mathbf{x}_{ik}, \mathbf{x}_{ij}) & \mathbf{C}(\mathbf{x}_{ik}) \end{bmatrix} \mathbf{J}_{\oplus}^T .$$

This method is easy to implement as a recursive algorithm. An equivalent method is to precompute the Jacobians of useful combinations of relationships such as the tail-to-tail combination by using the chain rule. Thus, the Jacobian of the tail-to-tail relationship, ${}_{\ominus}\mathbf{J}_{\oplus}$ is given by:

$$\begin{aligned} {}_{\ominus}\mathbf{J}_{\oplus} &\triangleq \frac{\partial \mathbf{x}_{jk}}{\partial (\mathbf{x}_{ij}, \mathbf{x}_{ik})} = \frac{\partial \mathbf{x}_{jk}}{\partial (\mathbf{x}_{ji}, \mathbf{x}_{ik})} \frac{\partial (\mathbf{x}_{ji}, \mathbf{x}_{ik})}{\partial (\mathbf{x}_{ij}, \mathbf{x}_{ik})} \\ &= \mathbf{J}_{\oplus} \begin{bmatrix} \mathbf{J}_{\ominus} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{1\oplus} \mathbf{J}_{\ominus} & \mathbf{J}_{2\oplus} \end{bmatrix} . \end{aligned}$$

Comparison will show that these two methods are symbolically equivalent, but the recursive method is easier to program, while pre-computing the composite Jacobians is more computationally efficient. Even greater computational efficiency can be achieved by making a change of variables such that the already computed mean estimate is used to evaluate the Jacobian, much as described earlier and in Appendix A.

It may appear that we are calculating first-order estimates of first-order estimates of ..., but actually this recursive procedure produces *precisely* the same result as calculating the first-order estimate of the composite relationship. This is in contrast to min-max methods which make conservative estimates at each step and thus produce *very* conservative estimates of a composite relationship.

If we now assume that the cross-covariance terms in the estimate of the covariance of the tail-to-tail relationship are zero, we get:

$$\mathbf{C}(\mathbf{x}_{jk}) \approx \mathbf{J}_{1\oplus} \mathbf{J}_{\ominus} \mathbf{C}(\mathbf{x}_{ij}) \mathbf{J}_{\ominus}^T \mathbf{J}_{1\oplus}^T + \mathbf{J}_{1\oplus} \mathbf{C}(\mathbf{x}_{ik}) \mathbf{J}_{1\oplus}^T$$

The Jacobians for six degree-of-freedom compounding and reversal relationships are given in Appendix A.

3.2.4 Extracting Relationships

We have now developed enough machinery to describe the procedure for estimating the relationships between objects which are in our map. The map contains, by definition, estimates of the locations of objects with respect to the world frame; these relations can be extracted directly. Other relationships are implicit, and must be extracted, using methods developed in the previous sections.

For any general spatial relationship among world locations we can write:

$$\mathbf{y} = \mathbf{g}(\mathbf{x}).$$

The estimated mean and covariance of the relationship are given by:

$$\begin{aligned} \hat{\mathbf{y}} &\approx \mathbf{g}(\hat{\mathbf{x}}), \\ \mathbf{C}(\mathbf{y}) &\approx \mathbf{G}_{\mathbf{x}} \mathbf{C}(\mathbf{x}) \mathbf{G}_{\mathbf{x}}^T . \end{aligned}$$

In our mobile robot example we will need to be able to estimate the relative location of one object with respect to the coordinate frame of another object in our map. In this case, we would simply substitute the tail-to-tail operation previously discussed for the function g.

$$\mathbf{y} = \mathbf{x}_{ij} = \ominus \mathbf{x}_i \oplus \mathbf{x}_j .$$

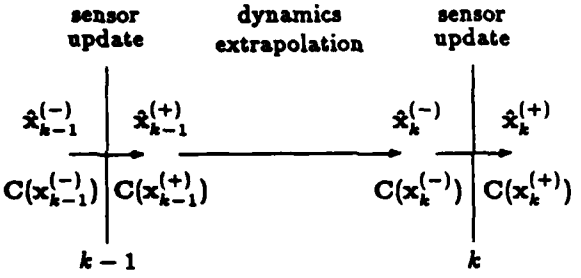


Figure 6: The Changing Map

4 Building the Map

Our map represents uncertain spatial relationships among objects referenced to a common world frame. Entries in the map may change for two reasons:

- An object moves.
- New spatial information is obtained.

To change the map, we must change the two components that define it — the (mean) estimate of the system state vector, $\hat{\mathbf{x}}$, and the estimate of the system variance matrix, $\mathbf{C}(\mathbf{x})$. Figure 6 shows the changes in the system due to moving objects, or the addition of new spatial information (from sensing).

We will assume that new spatial information is obtained at discrete moments, marked by states k. The update of the estimates at state k, based on new information, is considered to be instantaneous. The estimates, at state k, *prior* to the integration of the new information are denoted by $\hat{\mathbf{x}}_k^{(-)}$ and $\mathbf{C}(\mathbf{x}_k^{(-)})$, and *after* the integration by $\hat{\mathbf{x}}_k^{(+)}$ and $\mathbf{C}(\mathbf{x}_k^{(+)})$.

In the interval between states the system may be changing dynamically — for instance, the robot may be moving. When an object moves, we must define a process to extrapolate the estimate of the state vector and uncertainty at state $k-1$, to state k to reflect the changing relationships.

4.1 Moving Objects

Before describing how the map changes as the mobile robot moves, we will present the general case, which treats any processes that changes the state of the system.

The *system dynamics model*, or process model, describes how components of the system state vector change (as a function of time in a continuous system, or by discrete transitions).

Between state $k-1$ and k, no measurements of external objects are made. The new state is determined only by the process model, $\mathbf{f}$, as a function of the old state, and any control variables applied in the process (such as relative motion commands sent to our mobile robot). The process model is thus:

$$\mathbf{x}_k^{(-)} = \mathbf{f}\left(\mathbf{x}_{k-1}^{(+)}, \mathbf{y}_{k-1}\right), \tag{10}$$

where $\mathbf{y}$ is a vector comprised of control variables, $\mathbf{u}$, corrupted by mean-zero process noise, $\mathbf{w}$, with covariance $\mathbf{C}(\mathbf{w})$. That is, $\mathbf{y}$ is a noisy control input to the process, given by:

$$\mathbf{y} = \mathbf{u} + \mathbf{w}. \tag{11}$$

$$\hat{\mathbf{y}} = \mathbf{u}, \qquad \mathbf{C}(\mathbf{y}) = \mathbf{C}(\mathbf{w}).$$

Given the estimates of the state vector and variance matrix at state $k-1$, the estimates are extrapolated to state k by:

$$\hat{\mathbf{x}}_k^{(-)} \approx \mathbf{f}\left(\hat{\mathbf{x}}_{k-1}^{(+)}, \hat{\mathbf{y}}_{k-1}\right), \tag{12}$$

$$\mathbf{C}(\mathbf{x}_k^{(-)}) \approx$$

$$\mathbf{F}_{(\mathbf{x},\mathbf{y})} \begin{bmatrix} \mathbf{C}(\mathbf{x}_{k-1}^{(+)}) & \mathbf{C}(\mathbf{x}_{k-1}^{(+)}, \mathbf{y}_{k-1}) \\ \mathbf{C}(\mathbf{y}_{k-1}, \mathbf{x}_{k-1}^{(+)}) & \mathbf{C}(\mathbf{y}_{k-1}) \end{bmatrix} \mathbf{F}_{(\mathbf{x},\mathbf{y})}^T,$$

where,

$$\mathbf{F}_{(\mathbf{x},\mathbf{y})} = \begin{bmatrix} \mathbf{F}_\mathbf{x} & \mathbf{F}_\mathbf{y} \end{bmatrix} \triangleq \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{y})}{\partial(\mathbf{x},\mathbf{y})}\left(\hat{\mathbf{x}}_{k-1}^{(+)}, \hat{\mathbf{y}}_{k-1}\right)$$

If the process noise is uncorrelated with the state, then the off-diagonal sub-matrices in the matrix above are 0 and the covariance estimate simplifies to:

$$\mathbf{C}(\mathbf{x}_k^{(-)}) \approx \mathbf{F_X}\mathbf{C}(\mathbf{x}_{k-1}^{(+)})\mathbf{F_X}^T + \mathbf{F_Y}\mathbf{C}(\mathbf{y}_{k-1})\mathbf{F_Y}^T.$$

The new state estimates become the current estimates to be extrapolated to the next state, and so on.

In our example, only the robot moves, so the process model need only describe its motion. A continuous dynamics model can be developed given *a particular robot*, and the above equations can be reformulated as functions of time (see [Gelb, 1984]). However, if the robot only makes sensor observations at discrete times, then the discrete motion approximation is quite adequate.

When the robot moves, it changes its relationship, $\mathbf{x}_R$, with the world. The robot makes an uncertain relative motion, $\mathbf{y}_R = \mathbf{u}_R + \mathbf{w}_R$, to reach a final world location $\mathbf{x}'_R$. Thus,

$$\mathbf{x}'_R = \mathbf{x}_R \oplus \mathbf{y}_R.$$

Only a small portion of the map needs to be changed due to the change in the robot's location from state to state — specifically, the Rth element of the estimated mean of the state vector, and the Rth row and column of the estimated variance matrix. Thus, $\hat{\mathbf{x}}_{k-1}^{(+)}$ becomes $\hat{\mathbf{x}}_k^{(-)}$:

$$\hat{\mathbf{x}}_{k-1}^{(+)} = \begin{bmatrix} \hat{\mathbf{x}}_R \\ \hline \\ \\ \end{bmatrix}, \quad \hat{\mathbf{x}}_k^{(-)} = \begin{bmatrix} \hat{\mathbf{x}}'_R \\ \hline \\ \\ \end{bmatrix},$$

and, analogously, $\mathbf{C}(\mathbf{x}_{k-1}^{(+)})$ becomes:

$$\mathbf{C}(\mathbf{x}_k^{(-)}) = \left[\begin{array}{c|c|c} & \mathbf{B}'^T & \\ \hline \mathbf{B}' & \mathbf{A}' & \\ \hline & & \\ & & \\ & & \end{array}\right]$$

where:

$$\hat{\mathbf{x}}'_R \approx \hat{\mathbf{x}}_R \oplus \hat{\mathbf{y}}_R,$$

$$\mathbf{A}' = \mathbf{C}(\mathbf{x}'_R) \approx \mathbf{J}_{1\oplus}\mathbf{C}(\mathbf{x}_R)\mathbf{J}_{1\oplus}^T + \mathbf{J}_{2\oplus}\mathbf{C}(\mathbf{y}_R)\mathbf{J}_{2\oplus}^T,$$

$$\mathbf{B}'_i = \mathbf{C}(\mathbf{x}'_R, \mathbf{x}_i) \approx \mathbf{J}_{1\oplus}\mathbf{C}(\mathbf{x}_R, \mathbf{x}_i).$$

$\mathbf{A}'$ is the covariance matrix representing the uncertainty in the new location of the robot. $\mathbf{B}'$ is a row in the system variance matrix. The ith element is a sub-matrix — the cross-covariance of the robot's estimated location and the estimated location of the ith object, as given above. If the estimates of the two locations were not dependent, then that sub-matrix was, and remains 0. The newly estimated cross-covariance matrices are transposed, and written into the Rth column of the system variance matrix, marked by $\mathbf{B}'^T$.

4.2 New Spatial Information

The second process which changes the map is the update that occurs when new information about the system state is incorporated. New spatial information might be given, determined by sensor measurements, or even deduced as the consequence of applying a geometrical constraint. For example, placing a box on a table reduces the degrees of freedom of the box and eliminates the uncertainties in the lost degrees of freedom (with respect to the table coordinate frame). In our example, state information is obtained as prior knowledge, or through measurement.

There are two cases which arise when adding new spatial information about objects to our map:

- I: A new object is added to the map,
- II: A (stochastic) constraint is added between objects already in the map.

We will consider each of these cases in turn.

4.2.1 Case I: Adding New Objects

When a new object is added to the map, a new entry must be made in the system state vector to describe the object's *world* location. A new row and column are also added to the system variance matrix to describe the uncertainty in the object's estimated location, and the inter-dependencies of this estimate with estimated locations of other objects. The expanded system is:

$$\hat{\mathbf{x}}^{(+)} = \begin{bmatrix} \hat{\mathbf{x}}^{(-)} \\ \hline \hat{\mathbf{x}}_{n+1} \end{bmatrix}$$

$$\mathbf{C}(\mathbf{x}^{(+)}) = \left[\begin{array}{c|c} \mathbf{C}(\mathbf{x}^{(-)}) & \mathbf{B}^T \\ \hline \mathbf{B} & \mathbf{A} \end{array}\right],$$

where $\hat{\mathbf{x}}_{n+1}$, $\mathbf{A}$, and $\mathbf{B}$ will be defined below.

We divide Case I into two sub-cases: I-a, the estimate of the new object's location is *independent* of the estimates of other object locations described in the map; or I-b, it is *dependent* on them.

Case I-a occurs when the estimated location of the object is given directly in world coordinates — i.e., $\hat{\mathbf{x}}_{new}$ and $\mathbf{C}(\mathbf{x}_{new})$ — perhaps as prior information. Since the estimate is independent of other location estimates:

$$\mathbf{x}_{n+1} = \mathbf{x}_{new},$$

$$\hat{\mathbf{x}}_{n+1} = \hat{\mathbf{x}}_{new},$$

$$\mathbf{A} = \mathbf{C}(\mathbf{x}_{n+1}) = \mathbf{C}(\mathbf{x}_{new}), \qquad (13)$$

$$\mathbf{B}_i = \mathbf{C}(\mathbf{x}_{n+1}, \mathbf{x}_i) = \mathbf{C}(\mathbf{x}_{new}, \mathbf{x}_i) = 0.$$

where $\mathbf{A}$ is a covariance matrix, and $\mathbf{B}$ is a row of cross-covariance matrices, as before. $\mathbf{B}$ is identically 0, since the new estimate is independent of the previous estimates, by definition.

Case I-b occurs when the *world* location of the new object is determined as a function, $\mathbf{g}$, of its spatial relation, $\mathbf{z}$, to other object locations estimated in the map. The relation might be measured or given as prior information. For example, the robot measures the location of a new object relative to itself. Clearly, the uncertainty in the object's *world* location is correlated with the uncertainty in the robot's (world) location. For Case I-b:

$$\mathbf{x}_{n+1} = \mathbf{g}(\mathbf{x}, \mathbf{z}),$$

$$\hat{\mathbf{x}}_{n+1} = \mathbf{g}(\hat{\mathbf{x}}, \hat{\mathbf{z}}),$$

$$\mathbf{A} = \mathbf{C}(\mathbf{x}_{n+1}) = \mathbf{G}_{\mathbf{X}}\mathbf{C}(\mathbf{x})\mathbf{G}_{\mathbf{X}}^T + \mathbf{G}_{\mathbf{y}}\mathbf{C}(\mathbf{z})\mathbf{G}_{\mathbf{y}}, \quad (14)$$

$$\mathbf{B}_i = \mathbf{C}(\mathbf{x}_{n+1}, \mathbf{x}_i),$$

$$\mathbf{B} = \mathbf{G}_{\mathbf{X}}\mathbf{C}(\mathbf{x}).$$

We see that Case I-a is the special case of Case I-b, where estimates of t' e world locations of new objects are independent of the old state estimates and are given exactly by the measured information. That is, when:

$$\mathbf{g}(\mathbf{x}, \mathbf{z}) = \mathbf{z}.$$

4.2.2 Case II: Adding Constraints

When new information is obtained relating objects *already in the map*, the system state vector and variance matrix do not increase in size; i.e., no new elements are introduced. However, the old elements are *constrained* by the new relation, and their values will be changed.

Constraints can arise in a number of ways:

- A robot measures the relationship of a *known* landmark to itself (i.e., estimates of the world locations of robot and landmark already exist).

- A geometric relationship, such as colinearity, coplanarity, etc., is given for some set of the object location variables.

In the first example the constraint is noisy (because of an imperfect measurement). In the second example, the constraint could be absolute, but could also be given with a tolerance.

There is no mathematical distinction between the two cases; we will describe all constraints as if they came from measurements by *sensors* — real sensors or pseudo-sensors (for geometric constraints), perfect measurement devices or imperfect. A pseudo-sensor which measures "rectangular-ness" is discussed later in the example.

When a constraint is introduced, there are two estimates of the geometric relationship in question — our current best estimate of the relation, which can be extracted from the map, and the new sensor information. The two estimates can be compared (in the same reference frame), and together should allow some improved estimate to be formed (as by averaging, for instance).

For each sensor, we have a *sensor model* that describes how the sensor maps the spatial variables in the state vector into sensor variables. Generally, the measurement, $\mathbf{z}$, is described as a function, $\mathbf{h}$, of the state vector, corrupted by mean-zero, additive noise $\mathbf{v}$. The covariance of the noise, $\mathbf{C(v)}$, is given as part of the model.

$$\mathbf{z} = \mathbf{h(x)} + \mathbf{v}. \tag{15}$$

The *conditional* sensor value, given the state, and the *conditional covariance* are easily estimated from (15) as:

$$\hat{\mathbf{z}} \approx \mathbf{h}(\hat{\mathbf{x}}).$$

$$\mathbf{C(z)} \approx \mathbf{H_X C(x) H_X^T} + \mathbf{C(v)},$$

where:

$$\mathbf{H_X} \triangleq \frac{\partial \mathbf{h}_k(\mathbf{x})}{\partial \mathbf{x}} \left(\hat{\mathbf{x}}_k^{(-)}\right)$$

The formulae describe what values we *expect* from the sensor under the circumstances, and the likely variation; it is our current best estimate of the relationship to be measured. The actual sensor values returned are usually assumed to be conditionally independent of the state, meaning that the noise is assumed to be independent in each measurement, even when measuring the same relation with the same sensor. The actual sensor values, corrupted by the noise, are the second estimate of the relationship.

For simplicity, in our example we assume that the sensor measures the relative location of the observed object in Cartesian coordinates. Thus the sensor function becomes the tail-to-tail relation of the location of the sensor and the sensed object. described in Section 3.2.3. (Formally, the sensor function is a function of all the variables in the state vector, but the unused variables are not shown below):

$$\mathbf{z} = \mathbf{x}_{ij} = \ominus \mathbf{x}_i \oplus \mathbf{x}_j.$$

$$\hat{\mathbf{z}} = \hat{\mathbf{x}}_{ij} = \ominus \hat{\mathbf{x}}_i \oplus \hat{\mathbf{x}}_j.$$

$$\mathbf{C(z)} = {}_{\ominus}\mathbf{J}_{\oplus} \begin{bmatrix} \mathbf{C(x_i)} & \mathbf{C(x_i, x_j)} \\ \mathbf{C(x_j, x_i)} & \mathbf{C(x_j)} \end{bmatrix} {}_{\ominus}\mathbf{J}_{\oplus}^{\mathbf{T}} + \mathbf{C(v)}.$$

Given the sensor model, the conditional estimates of the sensor values and their uncertainties, and an actual sensor measurement, we can update the state estimate using the Kalman Filter equations [Gelb, 1984] given below, and described in the next section:

$$\hat{\mathbf{x}}_k^{(+)} = \hat{\mathbf{x}}_k^{(-)} + \mathbf{K}_k \left[\mathbf{z}_k - \mathbf{h}_k(\hat{\mathbf{x}}_k^{(-)})\right],$$

$$\mathbf{C}(\mathbf{x}_k^{(+)}) = \mathbf{C}(\mathbf{x}_k^{(-)}) - \mathbf{K}_k \mathbf{H_X C}(\mathbf{x}_k^{(-)}), \tag{16}$$

$$\mathbf{K}_k = \mathbf{C}(\mathbf{x}_k^{(-)}) \mathbf{H_X^T} \left[\mathbf{H_X C}(\mathbf{x}_k^{(-)}) \mathbf{H_X^T} + \mathbf{C(v)}_k\right]^{-1}.$$

4.2.3 Kalman Filter

The updated estimate is a weighted average of the two estimates, where the weighting factor (computed in the weight matrix K) is proportional to the prior covariance in the state estimate, and inversely proportional to the conditional covariance of the measurement. Thus, if the measurement covariance is large, compared to the state covariance, then $\mathbf{K} \rightarrow 0$, and the measurement has little impact in revising the state estimate. Conversely, when the prior state covariance is large compared to the *noise covariance*, then $\mathbf{K} \rightarrow \mathbf{I}$, and nearly the entire difference between the measurement and its expected value is used in updating the state.

The Kalman Filter generally contains a system dynamics model defined less generally than presented in (10); in the standard filter equations the process noise is additive:

$$\mathbf{x}_k^{(-)} = \mathbf{f}\left(\mathbf{x}_{k-1}^{(+)}, \mathbf{u}_{k-1}\right) + \mathbf{w}_{k-1} \qquad (17)$$

in that case $\mathbf{F_y}$ of (10) is the identity matrix, and the estimated mean and covariance take the form:

$$\hat{\mathbf{x}}_k^{(-)} \approx \mathbf{f}\left(\hat{\mathbf{x}}_{k-1}^{(+)}, \mathbf{u}_{k-1}\right), \qquad (18)$$

$$\mathbf{C}(\mathbf{x}_k^{(-)}) \approx \mathbf{F_X}\mathbf{C}(\mathbf{x}_{k-1}^{(+)})\mathbf{F_X}^T + \mathbf{C}(\mathbf{w}_{k-1}).$$

If the functions **f** in (17) and **h** in (15) are *linear* in the state vector variables, then the partial derivative matrices **F** and **H** are simply constants, and the update formulae (16) with (17), (15), and (18), represent the Kalman Filter [Gelb, 1984].

If, in addition, the noise variables are drawn from normal distributions, then the Kalman Filter produces the *optimal minimum-variance Bayesian estimate*, which is equal to the mean of the *a posteriori conditional density function* of **x**, given the prior statistics of **x**, and the statistics of the measurement **z**. No non-linear estimator can produce estimates with smaller mean-square errors.

If the noise does not have a normal distribution, *then the Kalman Filter is not optimal*, but produces the optimal *linear* estimate.

If the functions **f** and **h** are *non-linear* in the state variables, then **F** and **H** will have to be evaluated (they are not constant matrices). The given formulae then represent the Extended Kalman Filter, a sub-optimal non-linear estimator. It is one of the most widely used non-linear estimators because of its similarity to the optimal linear filter, its simplicity of implementation, and its ability to provide accurate estimates in practice.

The error in the estimation due to the non-linearities in **h** can be greatly reduced by iteration, using the Iterated Extended Kalman Filter equations [Gelb, 1984]:

$$\hat{\mathbf{x}}_{k,i+1}^{(+)} = \hat{\mathbf{x}}_k^{(-)}$$

$$+ \mathbf{K}_{k,i}\left[\mathbf{z}_k - \left(\mathbf{h}_k(\hat{\mathbf{x}}_{k,i}^{(+)}) + \mathbf{H_X}(\hat{\mathbf{x}}_k^{(-)} - \hat{\mathbf{x}}_{k,i}^{(+)})\right)\right],$$

$$\mathbf{C}(\mathbf{x}_{k,i+1}^{(+)}) = \mathbf{C}(\mathbf{x}_k^{(-)}) - \mathbf{K}_{k,i}\mathbf{H_X}\mathbf{C}(\mathbf{x}_k^{(-)}),$$

$$\mathbf{K}_{k,i} = \mathbf{C}(\mathbf{x}_k^{(-)})\mathbf{H_X}^T\left[\mathbf{H_X}\mathbf{C}(\mathbf{x}_k^{(-)})\mathbf{H_X}^T + \mathbf{C}(\mathbf{v}_k)\right]^{-1},$$

where:

$$\mathbf{H_X} \triangleq \frac{\partial \mathbf{h}_k(\mathbf{x})}{\partial \mathbf{x}}\left(\hat{\mathbf{x}}_{k,i}^{(-)}\right)$$

$$\hat{\mathbf{x}}_{k,0}^{(+)} \triangleq \hat{\mathbf{x}}_k^{(-)}.$$

Note that the original measurement value, **z**, and the prior estimates of the mean and covariance of the state, are used in each step of the iteration. The ith estimate of the state is used to evaluate the weight matrix, **K**, and is the argument to the non-linear sensor function, **h**. Iteration can be carried out until there is little further improvement in the estimate. The final estimate of the covariance need only be computed at the end of iteration, rather than at each step, since the intermediate system covariance estimates are not used.

5 Developed Example

The methods developed in this paper will now be applied to the mobile robot example in detail. We choose the world reference frame to be the initial location of the robot, without loss of generality. The robot's initial location with respect to the world frame is then the identity relationship (of the compounding operation), with no uncertainty.

$$\hat{\mathbf{x}} = [\hat{\mathbf{x}}_R] = [\mathbf{0}],$$

$$\mathbf{C}(\mathbf{x}) = [\mathbf{C}(\mathbf{x}_R)] = [\mathbf{0}].$$

Note, that the normal distribution corresponding to this covariance matrix (from (4)) is singular, but the limiting case as the covariance goes to zero is a dirac delta function centered on the mean estimate. This agrees with the intuitive interpretation of zero covariance implying no uncertainty.

Step 1: When the robot senses object #1, the new information must be added into the map. Normally, adding new information relative to the robot's position would fall under case I-b, but since the robot's frame is the same as the world frame, it falls under case I-a. The sensor returns the mean location and variance of object #1 ($\hat{\mathbf{z}}_1$ and $\mathbf{C}(\mathbf{z}_1)$). The new system state vector and variance matrix are:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{\mathbf{x}}_R \\ \hat{\mathbf{x}}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \hat{\mathbf{z}}_1 \end{bmatrix},$$

$$\mathbf{C}(\mathbf{x}) = \begin{bmatrix} \mathbf{C}(\mathbf{x}_R) & \mathbf{C}(\mathbf{x}_R,\mathbf{x}_1) \\ \mathbf{C}(\mathbf{x}_1,\mathbf{x}_R) & \mathbf{C}(\mathbf{x}_1) \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}(\mathbf{z}_1) \end{bmatrix}.$$

where $\mathbf{x}_1$ is the location of object #1 with respect to the world frame.

Step 2: The robot moves from its current location to a new location, where the relative motion is given by $\mathbf{y}_R$. Since this motion is also from the world frame, it is a special case of the dynamics extrapolation.

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{\mathbf{x}}_R \\ \hat{\mathbf{x}}_1 \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{y}}_R \\ \hat{\mathbf{z}}_1 \end{bmatrix},$$

$$\mathbf{C}(\mathbf{x}) = \begin{bmatrix} \mathbf{C}(\mathbf{x}_R) & \mathbf{C}(\mathbf{x}_R,\mathbf{x}_1) \\ \mathbf{C}(\mathbf{x}_1,\mathbf{x}_R) & \mathbf{C}(\mathbf{x}_1) \end{bmatrix} = \begin{bmatrix} \mathbf{C}(\mathbf{y}_R) & \mathbf{0} \\ \mathbf{0} & \mathbf{C}(\mathbf{z}_1) \end{bmatrix}.$$

We can now transform the information in our map from the world frame to the robot's new frame to see how the world looks from the robot's point of view:

$$\hat{\mathbf{x}}_{RW} = \ominus\hat{\mathbf{x}}_R,$$
$$\mathbf{C}(\mathbf{x}_{RW}) \approx \mathbf{J}_{\ominus}\mathbf{C}(\mathbf{x}_R)\mathbf{J}_{\ominus}^T.$$

$$\hat{\mathbf{x}}_{R1} = \ominus\hat{\mathbf{x}}_R \oplus \hat{\mathbf{x}}_1,$$
$$\mathbf{C}(\mathbf{x}_{R1}) \approx \mathbf{J}_{1\oplus}\mathbf{J}_{\ominus}\mathbf{C}(\mathbf{x}_R)\mathbf{J}_{\ominus}^T\mathbf{J}_{1\oplus}^T + \mathbf{J}_{1\oplus}\mathbf{C}(\mathbf{x}_1)\mathbf{J}_{1\oplus}^T.$$

Step 3: The robot now senses an object from its new location. The new measurement, $\mathbf{z}_2$, is of course, relative to the robot's location, $\mathbf{x}_R$.

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{\mathbf{x}}_R \\ \hat{\mathbf{x}}_1 \\ \hat{\mathbf{x}}_2 \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{y}}_R \\ \hat{\mathbf{z}}_1 \\ \hat{\mathbf{y}}_R \oplus \hat{\mathbf{z}}_2 \end{bmatrix},$$

$$\mathbf{C}(\mathbf{x}) = \begin{bmatrix} \mathbf{C}(\mathbf{x}_R) & \mathbf{C}(\mathbf{x}_R,\mathbf{x}_1) & \mathbf{C}(\mathbf{x}_R,\mathbf{x}_2) \\ \mathbf{C}(\mathbf{x}_1,\mathbf{x}_R) & \mathbf{C}(\mathbf{x}_1) & \mathbf{C}(\mathbf{x}_1,\mathbf{x}_2) \\ \mathbf{C}(\mathbf{x}_2,\mathbf{x}_R) & \mathbf{C}(\mathbf{x}_2,\mathbf{x}_1) & \mathbf{C}(\mathbf{x}_2) \end{bmatrix} = \begin{bmatrix} \mathbf{C}(\mathbf{y}_R) & \mathbf{0} & \mathbf{C}(\mathbf{y}_R)\mathbf{J}_{1\oplus}^T \\ \mathbf{0} & \mathbf{C}(\mathbf{z}_1) & \mathbf{0} \\ \mathbf{J}_{1\oplus}\mathbf{C}(\mathbf{y}_R) & \mathbf{0} & \mathbf{C}(\mathbf{x}_2) \end{bmatrix}.$$

where:

$$\mathbf{C}(\mathbf{x}_2) = \mathbf{J}_{1\oplus}\mathbf{C}(\mathbf{y}_R)\mathbf{J}_{1\oplus}^T + \mathbf{J}_{2\oplus}\mathbf{C}(\mathbf{z}_2)\mathbf{J}_{2\oplus}^T.$$

Step 4: Now, the robot senses object #1 again. In practice one would probably calculate the world

FOUR UNCERTAIN POINTS APPLYING THE RECTANGLE CONSTRAINT

Figure 7:

location of a new object, and only after comparing the new object to the old ones could the robot decide that they are likely to be the same object. For this example, however, we will assume that the sensor is able to identify the object as being object #1 and we don't need to map this new measurement into the world frame before performing the update.

The symbolic expressions for the estimates of the mean and covariance of the state vector become too complex to reproduce as we have done for the previous steps. Also, if the iterated methods are being used, there is no symbolic expression for the results.

Notice that the formulae presented in this section are correct for *any* network of relationships which has the same topology as this example. This procedure can be completely automated, and is very suitable for use in off-line robot planning.

As a further example of some of the possibilities of this stochastic map method, we will present an example of a geometric constraint — four points known to be arranged in a rectangle. Figure 7 shows the estimated locations of the four points with respect to the world frame, before and after introduction of the information that they are the vertices of a rectangle. The improved estimates are overlayed on the original estimates in the "after" diagram. We model the rectangle constraint as we would any other sensor (with mean-zero noise):

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) + v.$$

In this case, we need a pseudo-sensor which measures the "rectangularity" of four points — $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k, \mathbf{x}_l$, labeled counter-clockwise from the lower-right corner:

$$\mathbf{z} = \begin{bmatrix} x_i - x_j + x_k - x_l \\ y_i - y_j + y_k - y_l \\ (x_i - x_j)(x_k - x_j) + (y_i - y_j)(y_k - y_j) \end{bmatrix}.$$

The first two elements of $\mathbf{z}$ are zero when opposite sides of the closed planar figure represented by the four vertices are parallel; the last element of $\mathbf{z}$ is zero when the two sides forming the upper-right corner are perpendicular. Given four estimated points, the prior conditional value of $\mathbf{z}$ and the estimated covariance can be computed. The new information — the "measurement" returned by the pseudo-sensor — will be drawn from a distribution with mean 0 and covariance determined by how much tolerance in the "rectangularity" parameters is acceptable. In fact, if we are going to impose the constraint that the four points are precisely in a rectangle — i.e., there is no measurement noise, $\mathbf{C}(\mathbf{v}) = 0$ — then we can choose $\mathbf{h}$ to be *any* function which is zero only when the four points are in a rectangle. If, however, we wish to impose a *loose* rectangle constraint, we must formulate the function $\mathbf{h}$ such that $\mathbf{z}$ is a useful measure of *how* the four points fail to be rectangular.

6 Discussion and Conclusions

This paper presents a general theory for estimating uncertain relative spatial relationships between reference frames in a network of uncertain spatial relationships. Such networks arise, for example, in industrial robotics and navigation for mobile robots, because the system is given spatial information in the form of sensed relationships, prior constraints, relative motions, and so on. The theory presented in this paper allows the efficient estimation of these uncertain spatial relations. This theory can be used, for example, to compute *in advance* whether a proposed sequence of actions (each with known uncertainty) is likely to fail due to too much accumulated uncertainty; whether a proposed sensor observation will reduce the uncertainty to a tolerable level; whether a sensor result is so unlikely given its expected value and its prior probability of failure that it should be ignored, and so on. This paper extends the theory of state estimation to include information in the form of uncertain spatial relations between many different frames.

The estimation procedure makes a number of assumptions that are normally met in practice. These assumptions are detailed in the text, but the main assumptions can be summarized as follows:

- The angular errors are "small". This requirement arises because we linearize inherently nonlinear relationships. In Monte Carlo simulations[Smith, 1985], angular errors with a standard deviation as large as 5° gave estimates of the means and variances to within 1% of the correct values.
- Estimating only two moments of the probability density functions of the uncertain spatial relationships is adequate for decision making. We believe that this is the case since we will most often model a sensor observation by a mean and variance, and the relationships which result from combining many pieces of information become rapidly Gaussian, and thus are accurately modelled by only two moments.

The theory presented in this paper can be extended to adaptively improve the models it uses. For example, if the noise term in a camera model is too large, the observed errors will be smaller on average than expected. Adaptive filtering methods can be incorporated into the methods described to improve model estimates.

Although the examples presented in this paper have been solely concerned with *spatial* information, there is nothing in the theory that imposes this restriction. Provided that functions are given which describe the relationships among the components to be estimated, those components could be forces, velocities, time intervals, or other quantities in robotic and non-robotic applications.

Appendix A

Earlier in this paper we presented formulae for computing the resultant of two spatial relationships in two dimensions (three degrees of freedom). In three dimensions, there are six degrees of freedom: translations in x, y, z and three orientation variables: ϕ, θ, ψ. There are two common interpretations of these orientation variables—Euler angles and roll, pitch, and yaw, defined below.

Euler Angles

Euler angles are defined by:

$$Euler(\phi, \theta, \psi) = Rot(z, \phi)Rot(y', \theta)Rot(z'', \psi)$$

The head to tail relationship is then given by:

$$\mathbf{x}_3 = \begin{bmatrix} x_3 \\ y_3 \\ z_3 \\ \phi_3 \\ \theta_3 \\ \psi_3 \end{bmatrix} = \begin{bmatrix} \mathbf{T}_E \\ \mathbf{A}_E \end{bmatrix}$$

where $\mathbf{T}_E$ and $\mathbf{A}$ are defined by:

$$\mathbf{T}_E = \mathbf{R}_1 \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} atan2(a_{y_3}, a_{x_3}) \\ atan2(a_{x_3}\cos\phi_3 + a_{y_3}\sin\phi_3, a_{z_3} \\ atan2(-n_{x_3}\sin\phi_3 + n_{y_3}\cos\phi_3, -o_{x_3}\sin\phi_3 + o_{y_3}\cos\phi_3 \end{bmatrix}$$

where $\mathbf{R}_1$ is defined below and a_{x_3} etc. are the corresponding elements of the compound rotation matrix $\mathbf{R}_3$, defined by $\mathbf{R}_3 = \mathbf{R}_1\mathbf{R}_2$. Note that the inverse trignometric function $atan2$ is a function of two arguments, the ordinate y and the abscissa x. This function returns the correct result when either x or y are zero, and gives the correct answer over the entire range of possible inputs [Paul, 1981].

The Jacobian of this relationship, $\mathbf{J}$, is:

$$\mathbf{J} = \frac{\partial \mathbf{x}_3}{\partial(\mathbf{x}_1, \mathbf{x}_2)} = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{M} & \mathbf{R}_{Euler} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{K}_1 & \mathbf{0}_{3\times3} & \mathbf{K}_2 \end{bmatrix}$$

$$\mathbf{M} = \begin{bmatrix} -(y_3 - y_1) & (z_3 - z_1)\cos(\phi_1) & o_{x_1}x_2 - n_{x_1}y_2 \\ x_3 - x_1 & (z_3 - z_1)\sin(\phi_1) & o_{y_1}x_2 - n_{y_1}y_2 \\ 0 & -x_2\cos\theta_1\cos\psi_1 + y_2\cos\theta_1\sin\psi_1 - z_2\sin\theta_1 & o_{z_1}x_2 - n_{z_1}y_2 \end{bmatrix}$$

$$\mathbf{R}_1 = \begin{bmatrix} n_{x_1} & o_{x_1} & a_{x_1} \\ n_{y_1} & o_{y_1} & a_{y_1} \\ n_{z_1} & o_{z_1} & a_{z_1} \end{bmatrix} =$$

$$\begin{bmatrix} \cos\phi_1\cos\theta_1\cos\psi_1 - \sin\phi_1\sin\psi_1 & -\cos\phi_1\cos\theta_1\sin\psi_1 - \sin\phi_1\cos\psi_1 & \cos\phi_1\sin\theta_1 \\ \sin\phi_1\cos\theta_1\cos\psi_1 + \cos\phi_1\sin\psi_1 & -\sin\phi_1\cos\theta_1\sin\psi_1 + \cos\phi_1\cos\psi_1 & \sin\phi_1\sin\theta_1 \\ -\sin\theta_1\cos\psi_1 & \sin\theta_1\sin\psi_1 & \cos\theta_1 \end{bmatrix}$$

$$\mathbf{K}_1 = \begin{bmatrix} 1 & [\cos\theta_3\sin(\phi_3-\phi_1)]/\sin\theta_3 & [\sin\theta_2\cos(\psi_3-\psi_2)]/\sin\theta_3 \\ 0 & \cos(\phi_3-\phi_1) & \sin\theta_2\sin(\psi_3-\psi_2) \\ 0 & \sin(\phi_3-\phi_1)/\sin\theta_3 & [\sin\theta_1\cos(\phi_3-\phi_1)]/\sin\theta_3 \end{bmatrix}$$

$$\mathbf{K}_2 = \begin{bmatrix} [\sin\theta_1\cos(\phi_3-\phi_1)]/\sin\theta_3 & [\sin(\psi_3-\psi_2)]/\sin\theta_3 & 0 \\ \sin\theta_2\sin(\psi_3-\psi_2) & \cos(\psi_3-\psi_2) & 0 \\ [\sin\theta_1\cos(\phi_3-\phi_1)]/\sin\theta_3 & [\cos\theta_3\sin(\psi_3-\psi_2)]/\sin\theta_3 & 1 \end{bmatrix}$$

The inverse relation, $\mathbf{x}'$, in terms of the elements of the relationship $\mathbf{x}$, using the Euler angle definition, is:

$$\mathbf{x}' = \begin{bmatrix} x' \\ y' \\ z' \\ \phi' \\ \theta' \\ \psi' \end{bmatrix} = \begin{bmatrix} -(n_x x + n_y y + n_z z) \\ -(o_x x + o_y y + o_z z) \\ -(a_x x + a_y y + a_z z) \\ -\psi \\ -\theta \\ -\phi \end{bmatrix}$$

where n_x etc. are the elements of the rotation matrix $\mathbf{R}$ defined above.

The Jacobian of the inverse Euler relationship is:

$$\mathbf{J} = \frac{\partial \mathbf{x}'}{\partial \mathbf{x}} = \begin{bmatrix} -\mathbf{R}^T & \mathbf{N} \\ \mathbf{0}_{3\times3} & \mathbf{Q} \end{bmatrix}, \qquad \mathbf{Q} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix},$$

$$\mathbf{N} = \begin{bmatrix} n_y x - n_x y & -n_z x\cos\phi - n_z y\sin\phi + z\cos\theta\cos\psi & -[o_z x + o_y y + o_z z] \\ o_y x - o_x y & -o_z x\cos\phi - o_z y\sin\phi - z\cos\beta\sin\psi & -[n_z x + n_y y + n_z z] \\ a_y x - a_x y & -a_z x\cos\phi - a_z y\sin\phi + z\sin\theta & 0 \end{bmatrix}.$$

Roll, Pitch and Yaw Angles

Roll, pitch, and yaw angles are defined by:

$$RPY(\phi,\theta,\psi) = Rot(z,\phi)Rot(y',\theta)Rot(x'',\psi)$$

The Jacobian of the head-to-tail relationship, with roll, pitch, and yaw variables is given by:

$$\mathbf{J} = \frac{\partial \mathbf{x}_3}{\partial(\mathbf{x}_1,\mathbf{x}_2)} = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{M} & \mathbf{R}_{RPY} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{K}_1 & \mathbf{0}_{3\times3} & \mathbf{K}_2 \end{bmatrix}$$

$$\mathbf{M} = \begin{bmatrix} -(y_3 - y_1) & (z_3 - z_1)\cos(\phi_1) & a_{x_1} y_2 - o_{x_1} z_2 \\ x_3 - x_1 & (z_3 - z_1)\sin(\phi_1) & a_{y_1} y_2 - o_{y_1} z_2 \\ 0 & -x_2 \cos\theta_1 - y_2 \sin\theta_1 \sin\psi_1 - z_2 \sin\theta_1 \cos\psi_1 & a_{z_1} y_2 - o_{z_1} z_2 \end{bmatrix}$$

$$\mathbf{R}_1 = \begin{bmatrix} n_{x_1} & o_{x_1} & a_{x_1} \\ n_{y_1} & o_{y_1} & a_{y_1} \\ n_{z_1} & o_{z_1} & a_{z_1} \end{bmatrix} =$$

$$\begin{bmatrix} \cos\phi_1 \cos\theta_1 & \cos\phi_1 \sin\theta_1 \sin\psi_1 - \sin\phi_1 \cos\psi_1 & \cos\phi_1 \sin\theta_1 \cos\psi_1 + \sin\phi_1 \sin\psi_1 \\ \sin\phi_1 \cos\theta_1 & \sin\phi_1 \sin\theta_1 \sin\psi_1 + \cos\phi_1 \cos\psi_1 & \sin\phi_1 \sin\theta_1 \cos\psi_1 - \cos\phi_1 \sin\psi_1 \\ -\sin\theta_1 & \cos\theta_1 \sin\psi_1 & \cos\theta_1 \cos\psi_1 \end{bmatrix}$$

$$\mathbf{K}_1 = \begin{bmatrix} 1 & [\sin\theta_3 \sin(\phi_3 - \phi_1)]/\cos\theta_3 & [o_{x_3} \sin\psi_3 + a_{x_3} \cos\psi_3]/\cos\theta_3 \\ 0 & \cos(\phi_3 - \phi_1) & \cos\theta_1 \sin(\phi_3 - \phi_1) \\ 0 & [\sin(\phi_3 - \phi_1)]/\cos\theta_3 & [\cos\theta_1 \cos(\phi_3 - \phi_1)]/\cos\theta_3 \end{bmatrix}$$

$$\mathbf{K}_2 = \begin{bmatrix} [\cos\theta_2 \cos(\psi_3 - \psi_2)]/\cos\theta_3 & [\sin(\psi_3 - \psi_2)]/\cos\theta_3 & 0 \\ \cos\theta_2 \sin(\psi_3 - \psi_2) & \cos(\psi_3 - \psi_2) & 0 \\ [a_{x_1} \cos\phi_3 + a_{y_1} \sin\phi_3]/\cos\theta_3 & [\sin\theta_3 \sin(\psi_3 - \psi_2)]/\cos\theta_3 & 1 \end{bmatrix}$$

Note that for both definitions, the Jacobian has been simplified by the use of final terms (e.g. x_3, ψ_3). Since the final terms are computed routinely in determining the mean relationship, they are available to evaluate the Jacobian. Examination of the elements indicates the possibility of a singularity; as the mean values of the angles approach a singular combination, the accuracy of the covariance estimates using this Jacobian will decrease. Methods for avoiding the singularity during calculations are being explored.

References

Brooks, R. A. 1985. Visual Map Making for a Mobile Robot. *Proc. IEEE Int. Conf. Robotics and Automation.* St. Louis: IEEE, pp. 824-829.

Brooks, R. A. 1982. Symbolic Error Analysis and Robot Planning. *Int. J. Robotics Res.* 1(4):29-68.

Chatila, R. and Laumond, J-P. 1985. Position Referencing and Consistent World Modeling for Mobile Robots. *Proc. IEEE Int. Conf. Robotics and Automation.* St. Louis: IEEE, pp. 138-145.

Gelb, A. 1984. *Applied Optimal Estimation.* M.I.T. Press

Nahi, N. E. 1976. *Estimation Theory and Applications.* New York: R.E. Krieger.

Papoulis, A. 1965. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill.

Paul, R. P. 1981. *Robot Manipulators: Mathematics, Programming and Control.* Cambridge: MIT Press.

Smith, R. C., and Cheeseman, P. C. 1985. On the Representation and Estimation of Spatial Uncertainty. SRI Robotics Lab. Tech. Paper, and accepted for publication in *Int. J. Robotics Res.*.

Smith, R. C., *et al.* 1984. Test-Bed for Programmable Automation Research. Final Report-Phase 1, SRI International, April 1984.

Taylor, R. H. 1976. A Synthesis of Manipulator Control Programs from Task-Level Specifications. AIM-282. Stanford, Calif.: Stanford University Artificial Intelligence Laboratory.

Appendix E

DETERMINING AN OBJECT'S LOCATION IN A ROBOT HAND BY MEANS OF VISION

SRI International

DETERMINING AN OBJECT'S LOCATION IN A ROBOT HAND BY MEANS OF VISION

August 1, 1984

By: Eitan Zeiler
Electronic Research Engineer

On leave from
Armament Development Authority
Haifa, Israel

International Fellow
Robotics Laboratory
Advanced Technology Division

333 Ravenswood Ave. • Menlo Park, CA 94025
415 326-6200 • TWX: 910-373-2046 • Telex: 334-486

ABSTRACT

An object being grasped by a robot may not be precisely aligned with the robot's hand. What is worse, identical objects that are grasped repeatedly during an automatic assembly procedure may vary frequently in their in-hand location. for instance, even if identical pegs are grasped, discrepancies in their locations in the robot's hand will prevent proper insertion of the pegs into the designated fixture holes.

Practical methods are described for calculating misgrasp location relative to ideal grasp location without any need for the complexity of geometric modeling and computation. A least-squares-error method and converging process are employed. The latter consists of an iterative scheme of "measure (picture-taking), compute, and move." This is done to reduce gross misgraspings (30-mm translation in plane and 15° rotation about the normal) to relatively small deviations of fractional millimeters and degrees from a normal ideal-grasp alignment.

The algorithm relates changes in the geometric features of a binary image to changes in position (x,y) and orientation (rotation about the normal) of an object. During training, known positional and orientational perturbations are applied to a reference object (prototype) to establish the correspondence between these two categories of changes. The location of the prototype in the robot's hand is considered as a reference location. About fifteen geometric features of the object's windowed picture that are sensitive to positional and orientational changes are used. The misgrasp location of the trained object is computed in Cartesian (4 x 4 matrix) transformation form. This transformation, called DIFF, is used to compensate for the robot's fingertip location for each grasp, in world coordinates, such that the object will be located repeatedly and precisely in the same ideal grasp location. This is achieved simply by multiplying the fingertip training transformation by the DIFF transformation. As for the example above, it locates the pegs repeatedly for each grasp such that the robot will be able to insert each peg precisely into a hole.

CONTENTS

ILLUSTRATIONS

I INTRODUCTION

So far, the sensory capability of typical robot hands is not even close to that of human hands. A robot's hands, for example, cannot measure part slippage while actually holding a part. One must also consider the locational error of the robot hand servomechanism in object manipulation. This error combined with the relative error of the object held, may make it impossible for the robot's hand to mate that object with a second object in a known location (e.g., in a fixture). To overcome this problem, we must "measure" the location of the held object relative to other objects (e.g., to the fixture). One way to perform such measurement is to bring the grasped object to a fixed location ("viewpoint") under a camera, measure the object's binary-image features, and compare them with those in a reference location, thus obtaining the locational error of the object.

Although this approach is applicable to general error detection and correction within six degrees of freedom (x, y, z, O_z, A_y, T_x), this paper deals with error measurement and correction in only three degrees of freedom--x, y, and O_z. The results nevertheless have practical application because a typical robot hand consists of two parallel planes that constrain the errors in the grasped object to only three degrees of freedom, arbitrarily assumed to be along x, y, and O_z.

Our method minimizes the difference between the changes in the image features and the normalized (x, y, O_z) errors by using least-squares fitting. We use linear approximation for the relationship between (x, y, O_z) errors and the corresponding changes in the image features (area, perimeter, radii, movements, and so on). Therefore, our technique is iterative and is limited to small grasping errors (e.g., $\pm$ 30 mm in x or y and $\pm$ 15° in O_z).

II OBJECT LOCATION IN A ROBOT'S COORDINATE SYSTEM

Three coordinate frames of the robot system are defined in Figure 1--R (robot frame), H (hand frame), and C (camera frame)--such that T is translated (not rotated) relative to R and the principal ray of the camera is normal to the x-y plane of R. In Figure 2 we define a coordinate frame O attached to the object such that its x and y coordinates are parallel and collinear with x and y of the hand frame, respectively. We also define the z axis of the flange frame to be collinear with the y axis of the hand frame, and denote the distance between the flange frame origin and the hand tip by z hand.

Let (T_h) view, (T_{flange}) view, and T_{hand} denote the arm-to-hand, arm-to-flange, and flange-to-hand coordinate transformation [Paul (1981)],

$$(T_h)\text{view} = (T_{flange})\text{view} * T_{hand} \ .$$

Figure 3 shows five coordinate frames:

- Arm frame R.
- Hand reference frame H, defined during a training procedure (to be described later).
- Object reference frame O, also defined during the training procedure.
- Object frame O', which is attached to the grasped object in its actual, erroneous location.
- Hand frame H', which is reached after successive arm motions (to be described later) until frame O' converges upon and then overlaps frame O.

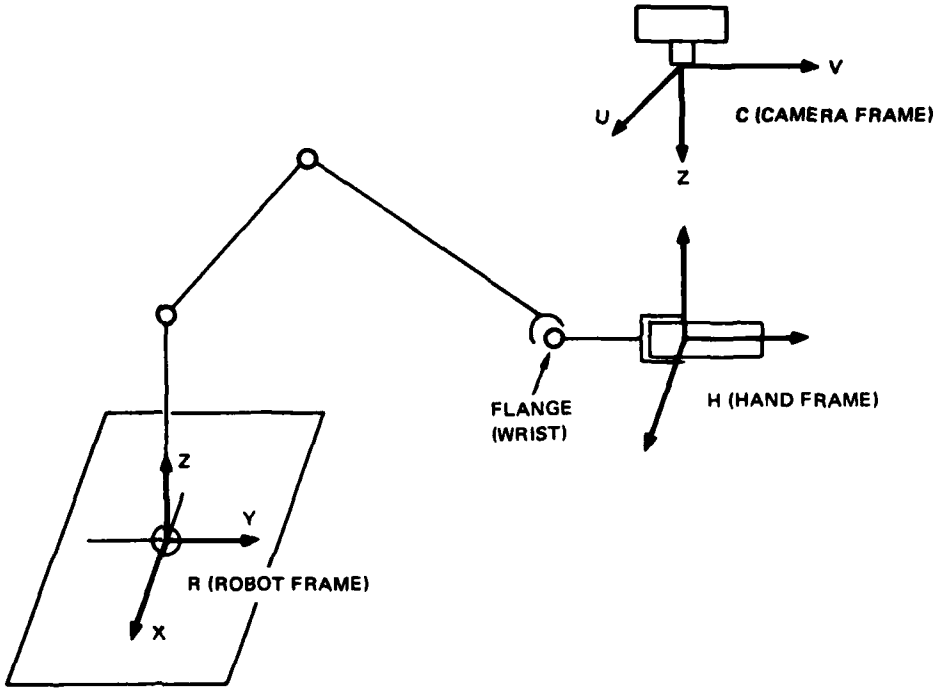


FIGURE 1 ROBOT, HAND, AND CAMERA COORDINATE FRAMES

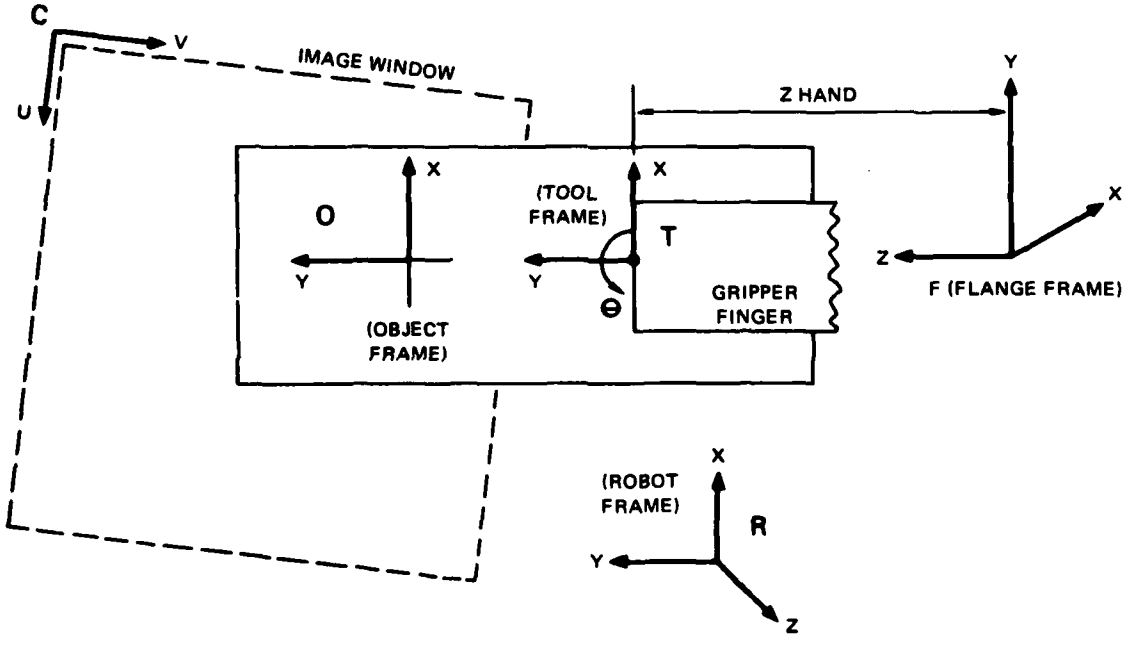


FIGURE 2 COORDINATE FRAMES AT ROBOT'S VISUAL-CHECK STATION

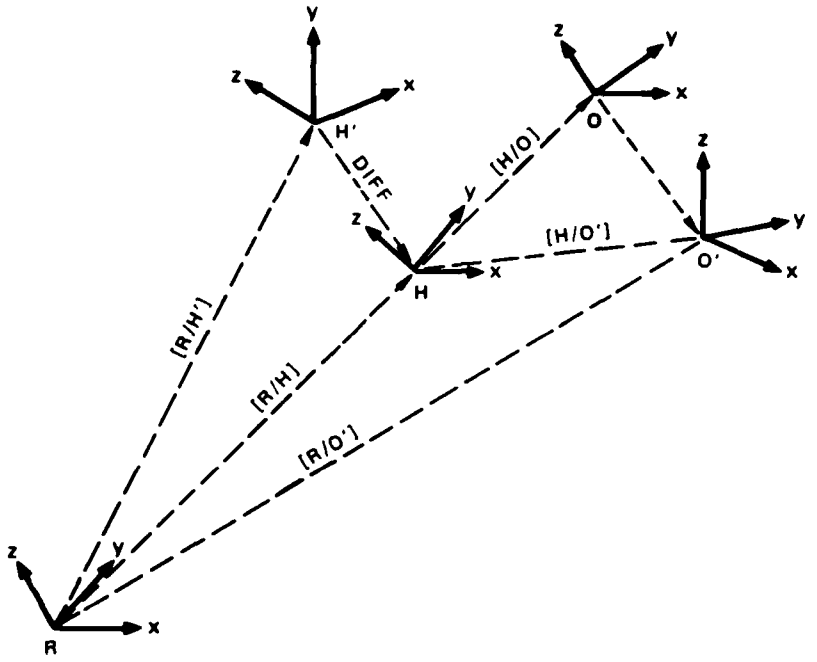


FIGURE 3 COORDINATE FRAMES OF R, H, O, AND H' AFTER ALIGNMENT OF FRAME O' WITH FRAME O

Denoting a general transformation from frame F1 to frame F2 by [F1/F2] [Smith and Nitzan (1983)], and denoting [H'/H] by DIFF, we obtain

$$\text{DIFF} = [\text{R/H'}]^{-1} * [\text{R/H}] \quad .$$

Since we have assumed that the errors in the grasped object are constrained to be only along x, y, or O_z, we obtain [Paul (1981)]

$$\text{DIFF} = \begin{vmatrix} \cos dO_z & -\sin dO_z & 0 & dx \\ \sin dO_z & \cos dO_z & 0 & dy \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

where dx, dy, and dO_z are the differences between the corresponding (x, y, O_z) values of frames H and H'.

DIFF values are computed for an array of (dx, dy, dO_z) values by grasping the object in different locations. These DIFF values are stored and then applied in the following way to correct for the corresponding grasping errors. The robot hand holding the object is moved to the reference location H underneath a sensor to the place where the training procedure has been performed. The sensor data are processed (as will be described later) to obtain the corresponding DIFF value. The resulting DIFF value is then used to offset the next hand transform, $[R/H]_{act}$, when an action would normally occur if there were no grasping errors, by multiplying $[R/H]_{act}$ by DIFF. For each grasp j we correct $[R/H]_{act}$ as follows:

$$[R/H]c_{actj} = [R/H]_{act} * DIFF_j \ .$$

III IMAGE PROCESSING

A. Global and Local Features

We use a binary visual sensor to measure the object's image features and correlate them with the value of DIFF. Two types of image features are distinguished: global [Gleason and Agin (1979)] and local [Bolles and Cain (1982)].

1. *Global Features*

Global features are those that can be obtained from the image when the entire object is visible and not touching any adjoining objects. In the calculation of global features, the object is assumed to be rigid. Global features are computed from the outline of an object, not from the regions enclosed within that outline.

Global features may be classified according to their dependence on the position and orientation of an object. A few examples are given below.

a. Independent Global Features

Some commonly used features that are independent of position and orientation are

- NCELLS--Number of pixels in the blob
- PERIMETER--Perimeter of the blob
- MAJOR--Length of major axis of the best-fit ellipse*
- RMIN--The minimum radius from the centroid to the perimeter.
- HOLERATIO--Ratio of the holes area to the total area.

---

*The best-fit ellipse is determined by finding an ellipse whose second moments are equal to those of the blob.

b. Dependent Global Features

Some commonly used features that depend on position and orientation are

- XMIN--The minimum x image coordinate of the blob
- XMAX--The maximum x image coordinate of the blob
- XPERIM--The fraction of the perimeter in x direction
- YPREIM--The fraction of the perimeter in y direction
- XCENT--The x coordinate of the blob's centroid
- YCENT--The y coordinate of the blob's centroid
- THETA--The angle of the major axis of the best-fit ellipse
- XDIFF--The width of the blob in the x direction
- YDIFF--The height of the blob in the y direction
- CGDIST--The absolute distance of the blob's centroid from the origin
- SIGXX--Summation of x squared
- SIGXY--Summation of x * y
- SIGYY--Summation of y squared.

Because the above features are independent of position and/or orientation, they are most useful in eliciting the information necessary for determining object location in robotic applications.

2. *Local Features*

Local features are image features that can be detected in a small window; e.g., small holes, corners (concave and convex), and the like. A matching method, called the Local-Feature-Focus Method, has been explored and developed to efficiently locate partially visible, two-dimensional objects [Bolles (1982)]. This approach is applicable to the recognition and location of complex industrial parts that may contain multiple occurrences of local features. The matching process is robust because it bases its decisions on unique clusters of features; it is also relatively fast because it concentrates on the most distinguished features, which are selected automatically.

B. Geometric Features for Determining Object's Location in a Robot's Hand

The problem we are dealing with here is that of determining, on the basis of an object's binary image, the location of that object as it is being held in a robot's hand. The use of global features is ruled out because the object is only partially visible. The use of local features is also excluded because there may not be enough of them to locate the object. To surmount these obstacles, we use the method described below.

The image seen by the TV camera includes an image of the robot's hand. Since the latter image is unwanted, we "cut it off" by defining a window that includes only a portion of the object's image; we then measure the global features of that portion. Since the object is only partially viewed, all these global features may vary as the object's location varies under a fixed camera and are thus sensitive to the object's position and orientation. Finally, we use a Jacobian (called sensitivity matrix) that ascertains the correspondence between the object's locational perturbations and the resulting changes in the values of the global features (see Section IV).

Incidently, the foregoing method may also be used to inspect the shape of an object in a fixed location. Any shape defect will manifest itself in a commensurate variation of the object's global features.

As will be described and explained in the next section, the sensitivity matrix is constructed experimentally by perturbing the object sequentially along each of the three degrees of freedom (x, y, and O_z), one at a time, and computing the blob's feature variations divided by the amount of the corresponding perturbation.

Fifteen global features have been selected according to two criteria:

- Maximum sensitivity to object location
- Feature independence.

Figure 4 shows the resulting 15 x 3 sensitivity matrix, classifies the selected 15 features according to their locational sensitivity, and lists their code numbers in the SRI vision module.

A mathematical model is required to convert the geometric-feature changes into locational changes. In addition, an algorithm for using this model is needed. The next section describes both the required mathematical model and the corresponding algorithm.

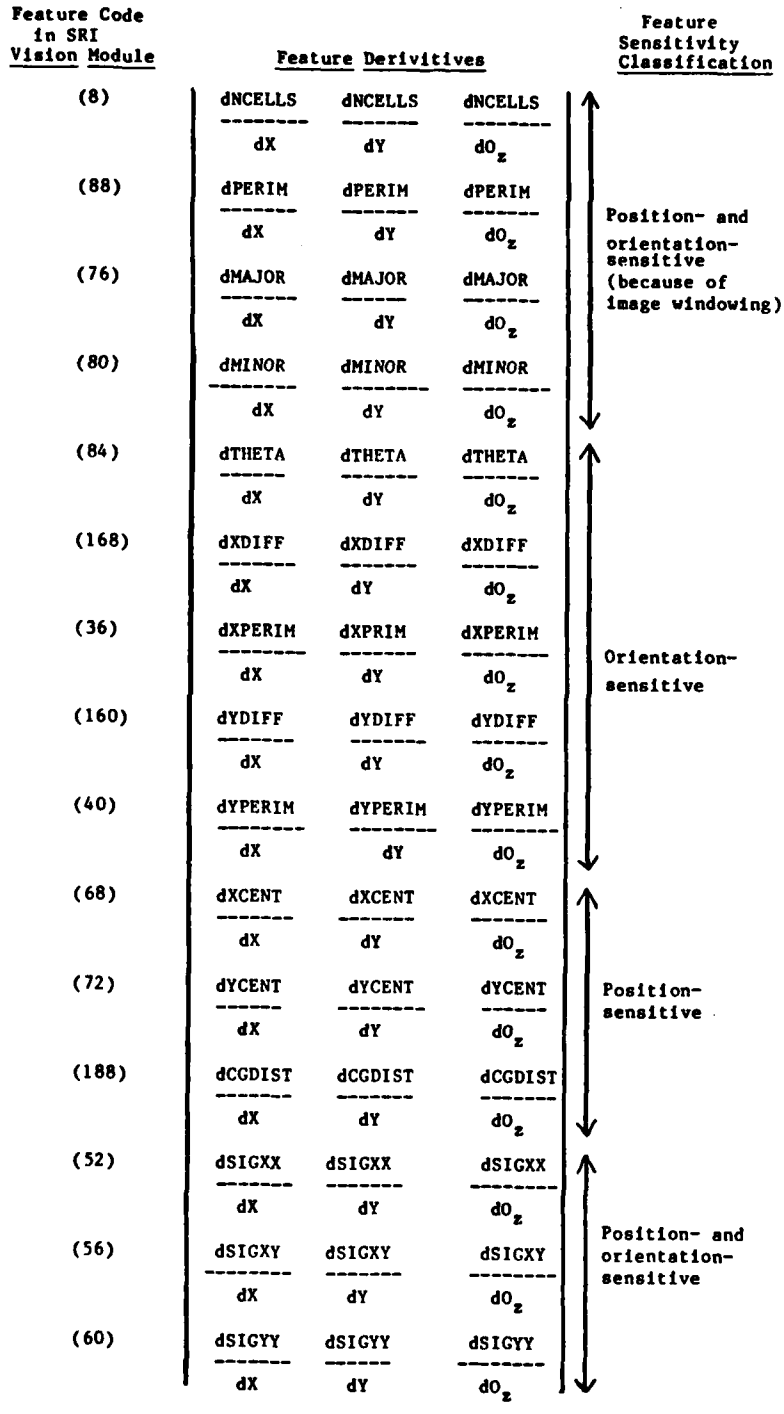


FIGURE 4 SENSITIVITY MATRIX OF GLOBAL FEATURES

IV MATHEMATICAL MODEL AND ALGORITHM

A practical method for computing the location of an object is described in this section. It converts geometric features of an object's image into an object location vector based on a linear mathematical model that is calibrated empirically. This technique does not require the complexity of geometric modeling and computation. The model is essentially linear for small deviations from an object's prototypical training location. The proposed algorithm employing this method enables the robot to attain the desired location so that the grasped object's location will coincide with that of the prototype.

A. Mathematical Model

Several geometric-feature changes in an object's image can be used to determine the three-dimensional position and orientation of that object. For sufficient information to be provided, the number of feature perturbations must be greater than or equal to the number of parameters to which the system is sensitive. In the case of the locational sensor, the number n of features measured must in general exceed 6, because a larger number of features is likely to increase the probability that every locational dimension will be well represented.

The n features that are dependent on position and orientation can be described as components of a feature eigenvector

$$\bar{f} = (f_1, f_2, f_3, \ldots\ldots\ldots f_n) \quad .$$

The location of an object in the robot's frame can be described by the vector

$$\bar{P} = (x, y, z, O_z, A_y, T_x) \quad .$$

The casual relationship between the feature eigenvector and the location vector may be expressed by

$$\bar{f} = T(\bar{P}) \quad ,$$

where T represent a nonlinear transfer function with multiple inputs and outputs. To provide an algorithm that will compute an object's location from the feature eigenvector, we have to invert the $T(\bar{P})$ form. We can use empirical calibration to produce a linear approximation of the function $T(\bar{P})$, then utilize it as a tool for controlling position and orientation.

We assume that, as a function of position and orientation, all the features will be continuous, as will the first derivatives. Let us denote the reference location by

$$\bar{P}0 = (x0, y0, z0, O_z0, A_y0, T_x0) \quad .$$

The corresponding location of the sample object defines the reference feature eigenvector as follows:

$$\bar{f}0 = T(\bar{P}0) = (f0_1, f0_2, f0_3, \ldots\ldots\ldots f0_n) \quad .$$

The nonlinear transformation $T(\bar{P})$ is dependent on many parameters, such as an object's geometric shape, the reference position of the sample object relative to camera coordinates, the kind of independent feature groups that have been computed, etc. Hence, as a first step of approximation, the location vector $\bar{P}$ will be obtained for small locational variations. Since we have restricted ourselves to small deviations, we can assume that the transformation of an object's location to its image features is linear; for a given $\bar{P}$ near $\bar{P}0$, we may use the approximation

$$T(\bar{P}) = T(\bar{P}0) + \left.\frac{dT}{dP}\right|_{\bar{P}0} * (\bar{P} - \bar{P}0) \quad .$$

Denoting the Jacobian matrix of the function T, evaluated at the reference location, by S we can write

$$\bar{f} = \bar{f}0 + S*(\bar{P} - \bar{P}0) \quad .$$

S symbolizes the sensitivity matrix (see Section III).

S is computed during the training (i.e., calibration) procedure, one column at time. This involves moving the robot's hand tip a small distance Dx, Dy, and Dz relative to the object's reference position and, in the same manner, rotating the hand tip by small angles DO_z, DA_y, and DT_x. Meanwhile, the variations in image features are computed and recorded, after which the following sensitivity matrix is constructed:

$$S = \begin{vmatrix} \frac{df_1}{dx} & \frac{df_1}{dy} & \frac{df_1}{dz} & \frac{df_1}{dO_z} & \frac{df_1}{dA_y} & \frac{df_1}{dT_x} \\ \frac{df_2}{dx} & \frac{df_2}{dy} & \frac{df_2}{dz} & \frac{df_2}{dO_z} & \frac{df_2}{dA_y} & \frac{df_2}{dT_x} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{df_n}{dx} & \frac{df_n}{dy} & \frac{df_n}{dz} & \frac{df_n}{dO_z} & \frac{df_n}{dA_y} & \frac{df_n}{dT_x} \end{vmatrix} \approx \begin{vmatrix} \frac{Df_1}{Dx} & \frac{Df_1}{Dy} & \frac{Df_1}{Dz} & \frac{Df_1}{DO_z} & \frac{Df_1}{DA_y} & \frac{Df_1}{DT_x} \\ \frac{Df_2}{Dx} & \frac{Df_2}{Dy} & \frac{Df_2}{Dz} & \frac{Df_2}{DO_z} & \frac{Df_2}{DA_y} & \frac{Df_2}{DT_x} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{Df_n}{Dx} & \frac{Df_n}{Dy} & \frac{Df_n}{Dz} & \frac{Df_n}{DO_z} & \frac{Df_n}{DA_y} & \frac{Df_n}{DT_x} \end{vmatrix}$$

Image features differ in their positional sensitivity according to object shape. Therefore, the use of redundant independent features would improve the response of the Jacobian S to a large variety of object shapes. It also is well known that, if the data points are independent and spread randomly, then the more of them there are (features, in our case), the better will be the estimation. It is the locational coordinates [Nahi (1976)] that we are estimating in this instance.

Because the number of image features exceeds the number of the required locational coordinates, of which there are three (x, y, O_z) in our application, S will became rectangular, thereby precluding the inverse of S. This means that the locational vector cannot be solved directly by the multiplication of $S^{-1} * \bar{f}$; instead, a pseudoinverse of S must be used. Letting

$$\bar{f}' = \bar{f} - \bar{f}0$$

$$\bar{P}' = \bar{P} - \bar{P}0$$

we are actually trying to solve the following overdetermined equation system, which does not have a unique solution:

$$\begin{vmatrix} & & \\ & S & \\ & & \end{vmatrix} * \begin{vmatrix} Px' \\ Py' \\ PO_z' \end{vmatrix} = \begin{vmatrix} f_1' \\ f_2' \\ \cdot \\ \cdot \\ \cdot \\ f_n' \end{vmatrix}$$

A location $\bar{P}c'$ can be found such that the computed product $S*\bar{P}c'$ fits the measured/computed feature $\bar{f}$ optimally. The residual vector $\bar{R}$ is defined by

$$\bar{R} = \bar{f}' - S * \bar{P}c' \quad ,$$

where $\bar{P}c'$is the computed location vector. Since minimizing the normal of the residual is the same as minimimizing the squares of $\bar{R}$ components, $\bar{R}$ is simply a least-squares fit of the locational parameters to the redundant image features, $(\bar{f}' - S * \bar{P}c')^T * (\bar{f}' - S * \bar{P}c') \text{---} > \min$.

The foregoing is like computing a plane that fits data containing more than three points as well as possible. The more data points are given, the better the estimated plane. The goodness of the fit is measured by the minimal sum of distances of the points from the plane. In our case, however we are dealing with image features rather than points in a Cartesian-coordinate system; the desired product here is a three-dimensional locational vector. The solution (see [Strang (1980)]) is

$$\bar{P}c' = (S^T * S)^{-1} * S^T * \bar{f}'$$

where $(S^T * S)^{-1} * S^T$ is called the pseudoinverse of the sensitivity matrix S. Note that $S^T * S$ is always a square matrix. For instance, given fifteen image features, S is a 15 x 3 matrix, S^T a 3 x 15 matrix, and $S^T * S$ a 3 x 3 matrix that can be inverted.

B. The Algorithm

1. *Sensor Range Extension*

The estimation method described in Section III-B is restricted to an extremely small range of measurable variations in the object's location. The higher the required accuracy, the smaller the range. This is because of the nonlinearity of the transformation function of the image-feature vector with respect to the locational vector. The smaller the changes, relative to a reference location, the better will the linear approximation fit the real nonlinear function T(P) around the reference point.

The algorithm proposed here for extending the measurable range of an object's location is valid as long as the tangents of $T(\bar{P})$ do not change directional polarity while the object moves beyond the linear region. Therefore, a procedure consisting of several steps (take a picture, compute feature changes, utilize the pseudoinverse to compute $\bar{P}'cj$, and move by $-\bar{P}'cj$--with j denoting the number of the steps) will cause a grasped object to move toward the prototype location. The stated condition, i.e., that the tangent outside the linear portion of $T(\bar{P})$ will not change polarity, guarantees that the object will move closer to the reference origin. On the other hand, while the object converges to the linear region of $T(\bar{P})$, the errors of the least-squares estimation become smaller. This is why the foregoing procedure converges such that a grasped object overlaps with the reference object.

The correction procedure would terminate when $\bar{P}'c$ coordinates (dx, dy, dO_z) are smaller than the required sensor threshold error. Figure 5 shows the result converging coordinates x-y to the origin. The experiment was done on an object that moves in a plane in x, y, O_z coordinates but only x-y was plotted.

Experiments show that, even when one of the three coordinates changes polarity, the correction procedure does converge. Experiments confirm that the coordinates are mutually dependent; when two coordinates move closer to the

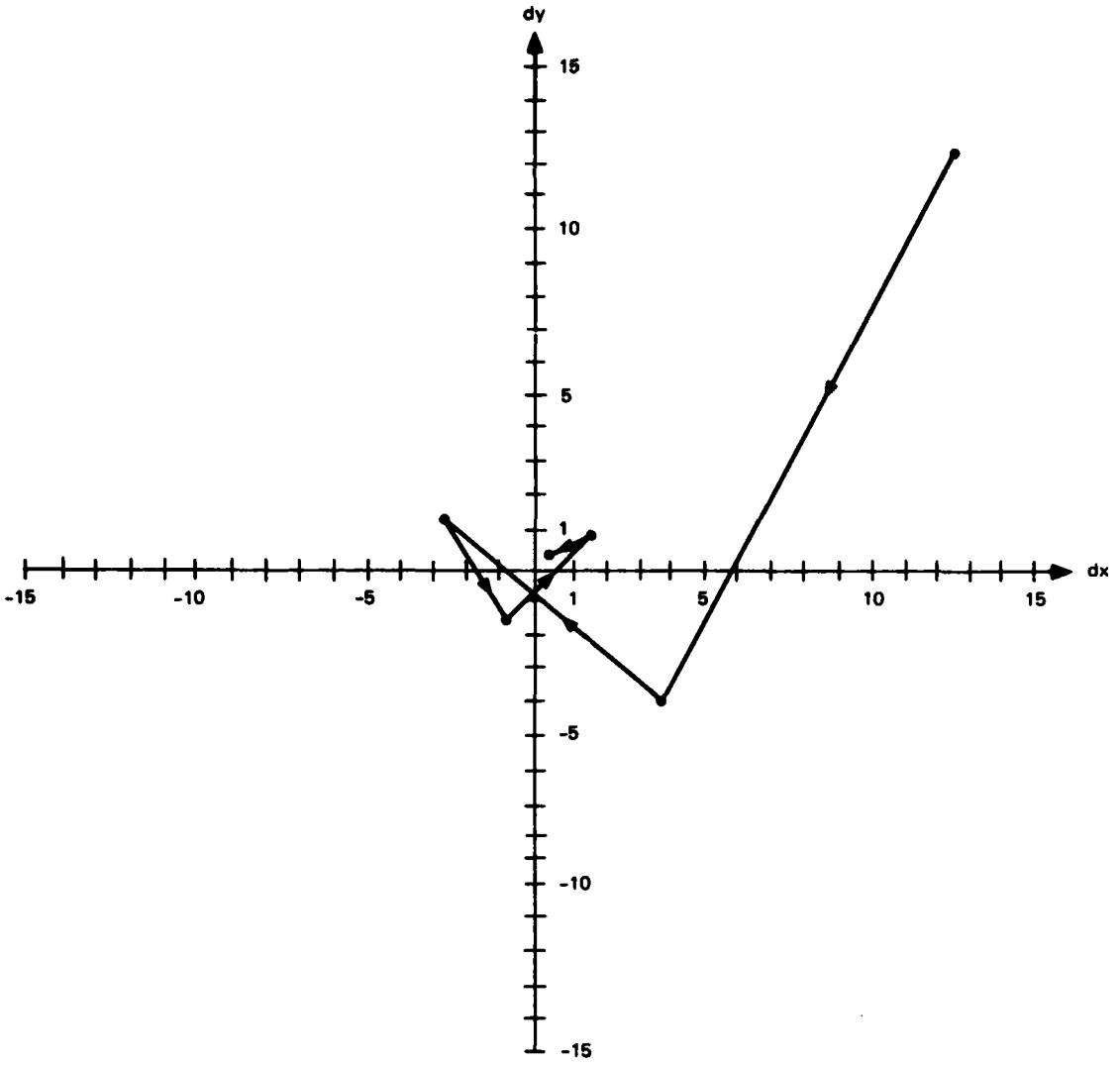


FIGURE 5 TYPICAL LOCATIONAL TRAJECTORIES OF A ROBOT HAND

origin, the third one does too. The number of iterations depends on the defined range and the accuracy required.

2. *The Algorithm*

The algorithm is illustrated diagrammatically in Figure 6. The diagram describes a section of automatic object location, as would be required in automatic assembly. First the robot grasps a reference object and a hand tip "actpoint" (actuation point) at the assembly station is defined. Second, the feature model must be calibrated, and the system trained for a reference object at the "viewpoint" where the camera sees the object. This involves computing and recording the sensitivity matrix, one column at a time, of feature derivatives with respect to each locational degree of freedom. Third, the calibrated relation between features and location is inverted according to the least-squares criterion; in other words, the pseudoinverse of matrix S is computed. Fourth, the robot grasps an object, moves to the "viewpoint," and the feature vector $\bar{f}'=\bar{f}-\bar{f}0$ is computed. Then the pseudoinverse of the sensitivity matrix multiplies the feature vector to provide locational coordinates. Finally, the robot is instructed to move the object by $-\bar{P}c'$ toward the reference location, as was defined in the training procedure.

Several iterations of picture-taking, computation of $\bar{P}c$, and moves by $-\bar{P}c$ may be performed until each of the object's coordinates is close enough to the reference location to be under the error threshold. When the robot's hand tip is in this location, the controller computes its transformation DIFF relative to the reference hand tip location determined during training. The last step is to instruct the robot to move to the "actpoint" location that compensates for misgrasp; i.e., the hand tip is moved to the (T_h) actpoint*DIFF location. (To compute DIFF, we use a routine called HERE VIEWPOINT:DIFF, which is a part of VAL-II, the program language of the robot controller.) Finally, the system is ready to grasp one more identical object.

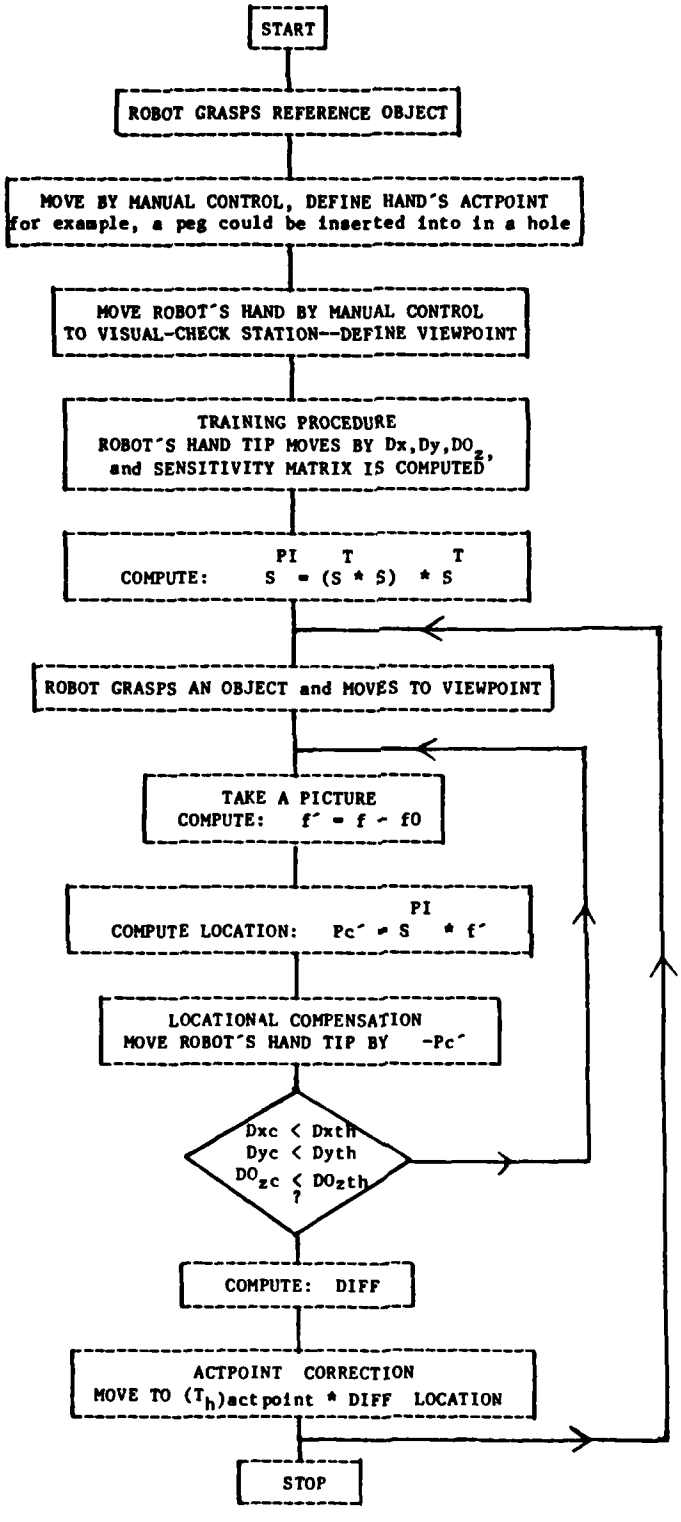


FIGURE 6 LOCATIONAL-CONTROL ALGORITHM

V SYSTEM DESCRIPTION AND DEMONSTRATION

A. System Description

1. *Introduction*

The subsystem we used for our experimental demonstration is part of a hierarchical, programmable assembly system. The latter consists of functional modules, each including a major device (e.g., a robot arm) or a sensor (e.g., a vision processor), as well as auxiliary devices (e,g.,an end-effector, such as a gripper). Each module is controlled by an LSI-11 computer, which stores reflex, bootstrap, and program routines for carrying out that module's functions. These computers are connected with one another and with the main system computer by means of a fast 10-MHz Ethernet communications network, using a single-coaxial-cable bus [Smith and Nitzan (1983)].

Figure 7 uses a block diagram to describe the assembly system; some modules that we did not use are shown within dashed lines. Our subsystem included one PUMA robot with its controller, an SRI vision module with a black-and-white graphic-display terminal (TEKTRONIX 4014-1), and an oscilloscope as an x-y image monitor. The real-time system controller is a DEC VAX-11/730, and the multiuser software-development system is implemented by a VAX-11/750. The high-level programming language used is "C"--developed and run on the VAX under the UNIX operating system. The "C" programs operate driver routines that enable the software functions to run on the SRI vision module or the PUMA controller.

Figures 8 and 9 show the part of the assembly system used by us. Figure 8 shows the manipulator module, including the PUMA robot and its end-effector, and the PUMA controller with its module computer mounted under the arm's supporting stand. The terminal depicted is connected directly to the robot's

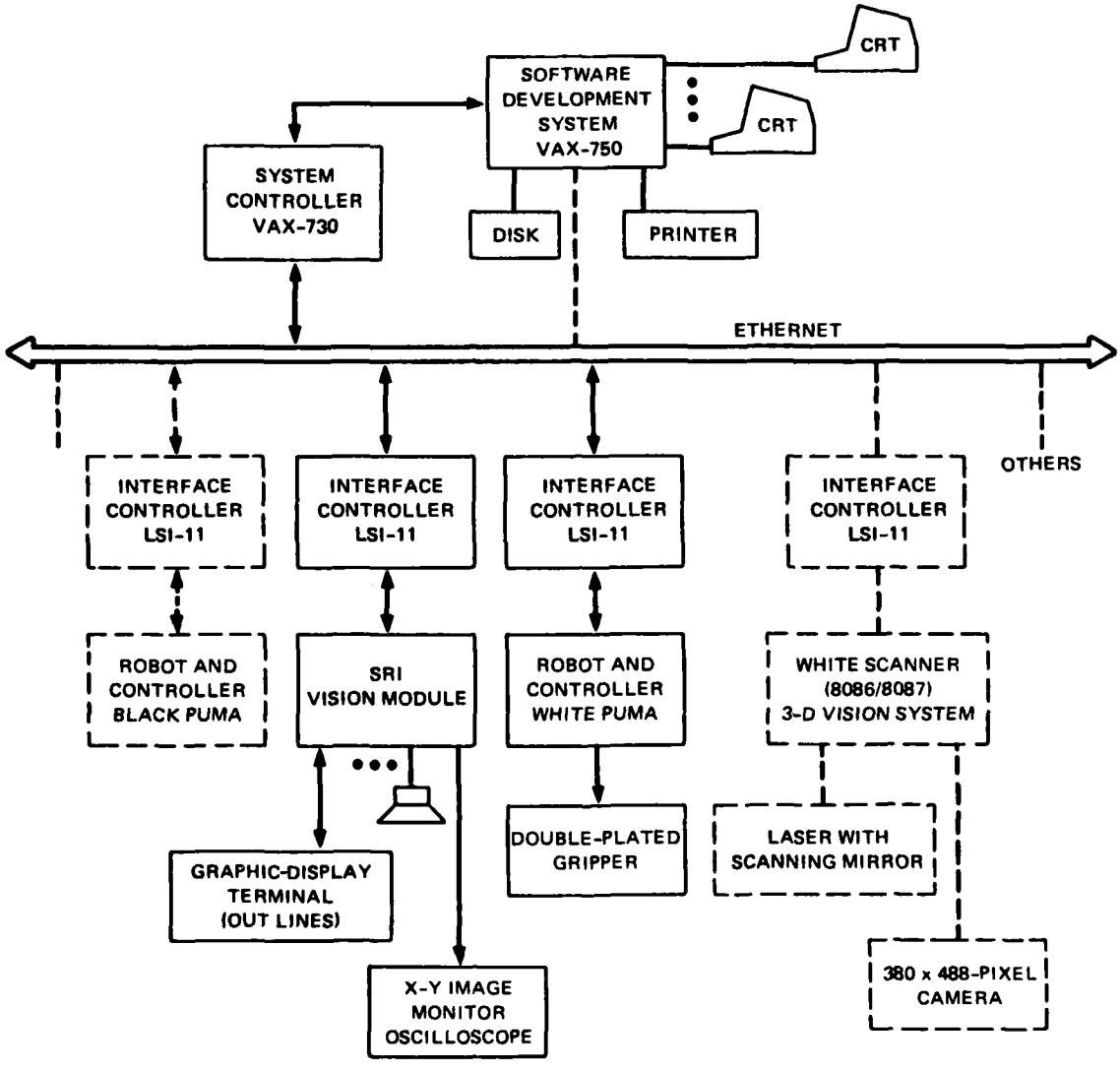


FIGURE 7 BLOCK DIAGRAM OF ASSEMBLY SYSTEM

controller and controls the robot by running VAL-II programs. This temporary connection was used for debugging. The control box on the terminal side is used for manual control of the robot and for the teaching mode. Figure 8 also shows a black table beneath the robot's hand that has a grid of holes; the vision module is mounted under the supporting stand of the grid table. Two of the three lamps we used and the TV camera are shown suspended from the ceiling. Figure 9 contains the terminal that is connected to the VAX-750 and from which the entire system is operated through the Ethernet bus, as well as the graphic-display terminal and the oscilloscope that serves as an x-y image monitor.

2. *Module Types*

The module computer contains a processor, network interface cards, memory, and input/output interface cards for the analog and digital signals from and to the auxiliary sensors and devices of a module. The specific modules we employed are described briefly below.

a. Manipulator Module

The manipulator module consist of a Unimation PUMA 560 robot and end-effector. The latter consists of a six-axis force/torque sensor mount on the wrist(not used by us) and a pneumatic two-fingered (flat plates) hand.

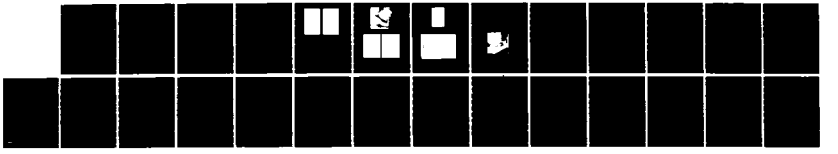
The PUMA controller is equipped with a VAL-II program that controls the arm. The module computer provides a means for controlling the hand, reading sensors therein as well as in the wrist, and for moving the arm indirectly by communication with the PUMA controller.

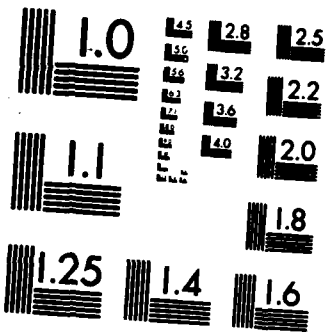
Here are a few examples of VAL-II functions used by us:

- WHERE (result)--Returns the location of the robot's hand tip.
- MOVE (location)--Moves to the specific location.

AD-A173 121 DEVELOPMENT SYSTEM FOR FLEXIBLE ASSEMBLY SYSTEM(U) SRI 2/2
INTERNATIONAL MENLO PARK CA R C SMITH ET AL. AUG 86
AFOSR-TR-86-1001 F49620-84-K-0007

UNCLASSIFIED F/G 9/2 NL





- HERE (location variable)--Defines the value of a transformation to be equal to the robot's current location. In our application, the "viewpoint" transformation is defined; then HERE VIEWPOINT:DIFF computes the right-most transformation DIFF. DIFF defines the current location relative to the "viewpoint" location in 4 x 4 transformation form.

b. Binary-Vision Module

The binary-vision module includes an SRI vision module [Gleason and Agin (1979)] with 128 x 128-element solid-state camera (one could attach up to four cameras), an LSI-11 computer module, and three lamps for high-intensity illumination. The camera is mounted on the ceiling, from which it sees the robot's hand-tip. A preprocessor in the SRI vision module divides the camera video signal into binary (either black or white) data. The LSI-11 computer of the SRI vision module includes an entire library of vision subroutines. The heart of image processing in the vision module is the connectivity analysis routine.

Two of the most important binary vision-module functions used by us are described as follows:

Picture (BlobCnt)

> Take a picture, perform a connectivity analysis of the image, and return the number of connected regions (blob) and the blob centroid (BlobCnt) that is computed by accumulating the first moments of the area of the blob about the x and y axes.

GetFeature(BlobN, FeatN, Result)

> Return the value of a blob feature (indicated by the index FeatN) of a selected blob (indicated by the index BlobN). Examples of such features are blob area, perimeter length, and moments.

We have used the capability of the vision module to specify a rectangular "window" in an image, outside which data are ignored. The purpose of our "windowing" application is to view the object only partially, ignoring the part that touches the gripper.

A standard oscilloscope with external x, y, and z inputs is used as an x-y monitor of the binary-image input to the vision processor. A graphic-display terminal, a TEKTRONIX 4014-1, is used to display processed images as outlines. We used a 75-mm, 1:1.4 lens. The distance from camera to viewpoint was about one meter, and one pixel of a 128 x 128 array was equivalent to about 0.4 mm.

B. Experimental Demonstration

1. *Utilizing Precise X-Y-Θ Manual Table*

Our first experiment was to run the proposed algorithm; instead of the manipulator module, however, we used a precise x-y-θ manual table. The object was placed on the table, entirely visible to the camera. The table has an error of 0.001" for translation, and 0.01° for rotation.

We trained the system (construction of the sensitivity matrix) for a translation of Dx = Dy = 0.1" and a rotation of $DO_z = 2°$. The technique was to subtract from the object's current location the xc, yc, O_zc computed by the system--in other words, to compute (P)j-(Pc)j+1 manually. This procedure was repeated until it converged to the origin. The accuracy we achieved was 0.005" (0.1mm) in x , y, and 0.4° in θ, within a range of 30 mm in x, y, and 16° in θ. The compute-move process was terminated within 4 to 5 iterations.

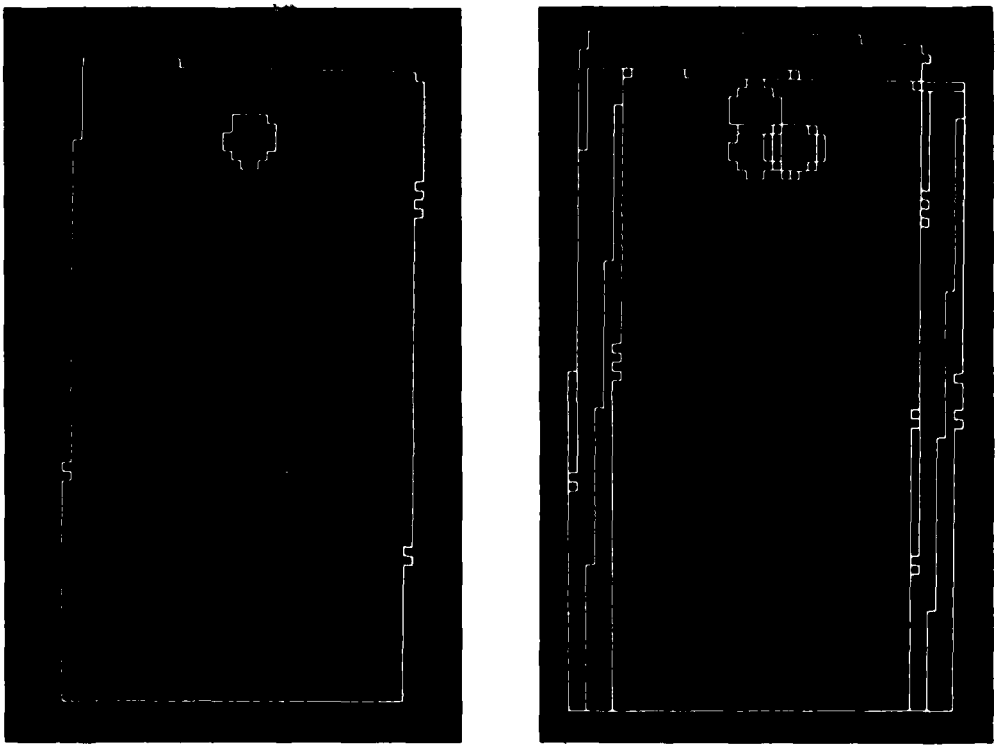
2. *Determining Object Location in Robot Hand*

Figure 6 diagrammatically illustrates the sensor algorithm, including the training procedure, the process of computing location, and the processing of the correction transformation DIFF. A special software package implementing that algorithm was written. This section shows a series of pictures taken of one among many experiments performed, recording the sensor processing step by step.

We first placed a sample object (workpiece) manually in the robot's hand; this was by definition the reference grasp (ideal grasp). Figure 10 shows the location of the robot's hand tip; i.e., where the camera sees the grasped object. This is where the viewpoint transformation of the hand tip is defined (computed). Figure 11 shows how the robot is moved manually such that the workpiece touches the edge of the peg; it is at this point that the actpoint transformation is defined.

Figure 12a reveals part of the workpiece at its reference location, viewed through a "window" as it is displayed on the graphic monitor. Figure 12b shows four outlines of the workpiece at its training locations. The training translation of each coordinate Dx and Dy is 3 mm and the rotation DO_z is 2°, relative to the reference location.

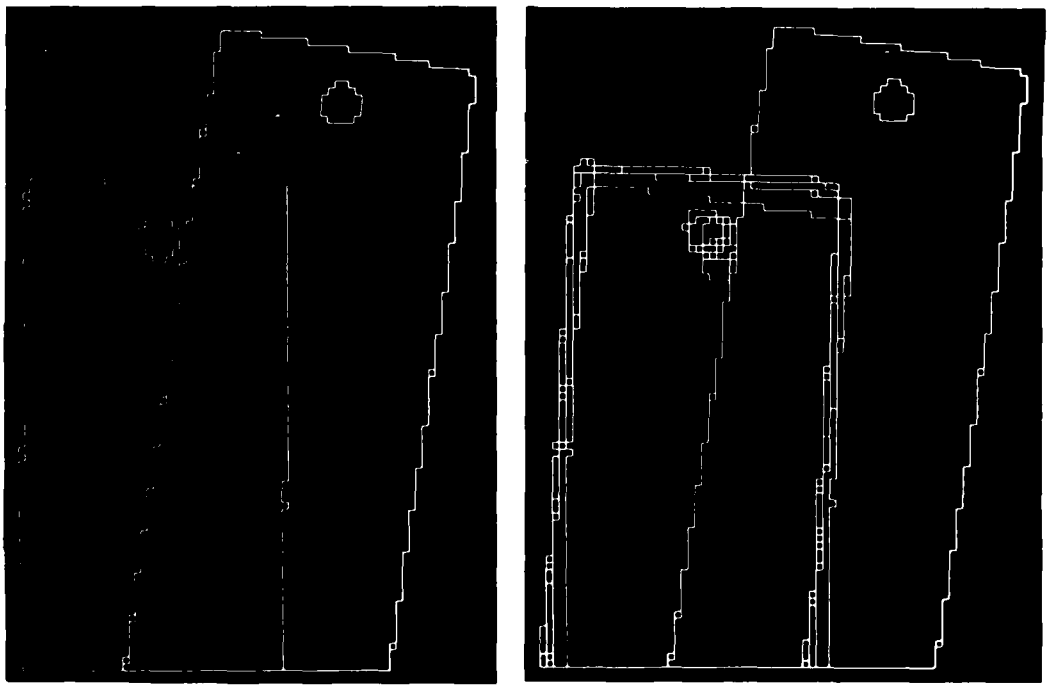
Figure 13 shows how the object is moved manually in a robot's hand to an unknown location to simulate a misgrasp of an object. Figure 14a shows such a misgrasp compared with the ideal grasp at a reference location. Figure 14b shows five image outlines that are four compute-move iterations of an object's locational convergence process. It indicates clearly how the robot moves until a grasped object is located at the reference location, with the number of iterations dependent on the accuracy required. Figure 15 illustrates the final location of a grasped object, compared with the reference location. In this specific experiment the final location error $\overline{Pc}$ was $\overline{Pc} = (Dx, Dy, DO_z) = (0.088mm, -0.7mm, -0.48mm)$. The terminal connected to the robot's controller shows (Figure 16) the Cartesian coordinates (x,y,z,O,A,T), which coincide with the blob's outlines in Figure 14b, in the robot frame. Figure 16 shows the intermediate locations until convergence at a hand tip location labeled targ5, the training locations targx, targy, targO, and the reference location viewpoint.



(a) (b)

FIGURE 12 WORKPIECE AT REFERENCE LOCATION (a); TRAINING LOCATIONS (b)

FIGURE 13 MISGRASP OF AN OBJECT



(a) OBJECT MISGRASPED COMPARED WITH SAMPLE OBJECT GRASP

(b) COMPUTE-MOVE ITERATION TOWARDS THE REFERENCE LOCATION

FIGURE 14 OUTLINES OF THE OBJECT'S IMAGE

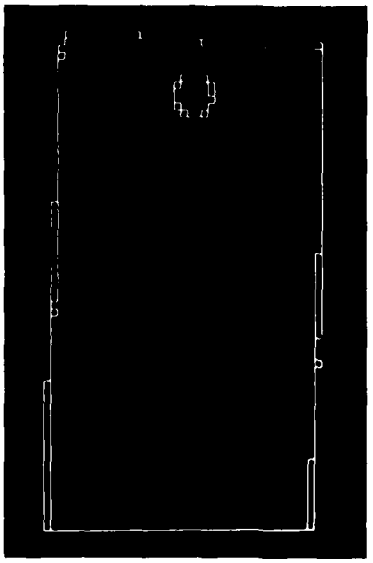


FIGURE 15 FINAL LOCATION OF MISGRASPED OBJECT
COMPARED WITH REFERENCE OBJECT

```
.
.
.
.
.
.listl
            X/Jt1     Y/Jt2     Z/Jt3     O/Jt4     A/Jt5     T/Jt6
diff         6.81     -1.50    -14.75     84.320   -82.667     5.350
targ      -692.81     54.97    -101.09    -87.523    -0.022    95.224
targ1     -694.44     56.41    -101.28    -86.622     0.016    95.136
targ2     -694.91     58.66    -100.91    -83.903    -0.044    95.092
targ3     -694.00     59.59    -100.94    -83.403    -0.044    95.059
targ4     -693.91     60.69    -100.97    -82.397    -0.066    94.944
targ5     -693.41     61.31    -101.38    -82.502    -0.027    94.927
targo     -708.16     68.00    -100.97    -91.813    -0.022    95.257
targx     -711.16     68.00    -100.97    -89.813    -0.022    95.257
targy     -708.16     71.00    -100.97    -89.813    -0.022    95.257
tool1        0.00      0.00     106.00     90.000   -90.000     0.000
viewpoint -708.16     68.00    -100.97    -89.813    -0.022    95.257
.do move viewpoint:diff
.
.
.
.
```

FIGURE 16 CARTESIAN COORDINATES OF HAND TIP IN ROBOT'S FRAME

When the process to minimize Pc terminates, the correction transformation DIFF is computed (shown on the terminal of the robot controller diff in Figure 16). In the experiment the computer instructs the robot to move from arbitrary location to viewpoint*DIFF location (Viewpoint ≡ [R/H], the result shows almost the same overlapped outlines as in Figure 15. Figure 17 shows the robot holding an object at the hand tip actpoint*DIFF location (actpoint ≡ $[R/H]_{act}$. We can see clearly here that the edge of the object (workpiece) touches the edge of the peg just as the trained sample or reference object had done.

The accuracy of the system is limited to that of PUMA, which is 1 mm for translation and 1 ° for rotation. Using the precise x-y-θ table, we achieved accuracy of 0.1 mm, 0.4 ° (see Section V-B1).

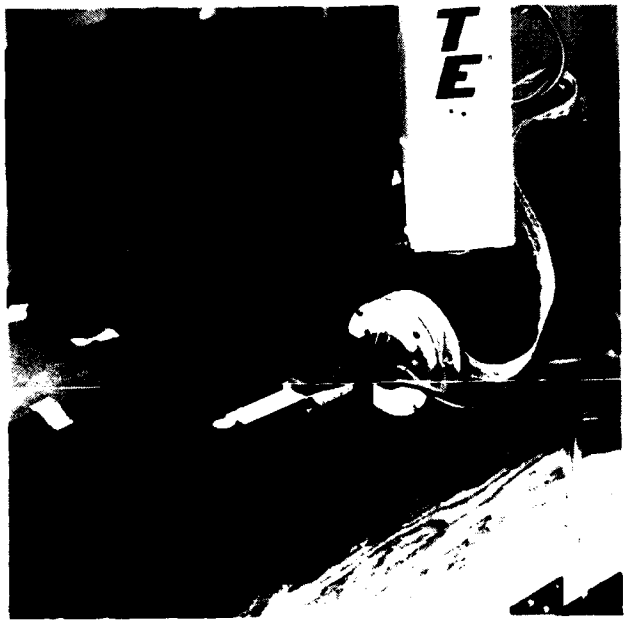


FIGURE 17 ROBOT'S HAND TIP AT ACTPOINT:DIFF — ACTPOINT CORRECTED LOCATION

VI SUMMARY AND FUTURE WORK

A. Summary

A practical method and an algorithm for computing an object's location in a robot's hand have been presented. The method converts image features of that object into an object location vector, doing this on the basis of a least-squares-error estimation calibrated empirically during a training procedure.

Application of the algorithm was implemented for object location in a plane by establishing three degrees of freedom--i.e., x-y for translation and O_z for rotation. That application is practical for a robot with a hand that has two flat fingers. The method can determine object location even when only a part of the object is visible to the camera and, moreover, this ability is independent of the object's shape. it is not like the method of maximal cliques, which can ascertain the location of incomplete images, but depends on the kind and number of subblobs that are visible to the camera. Furthermore, the maximal-clique approach is very expensive in terms of processing time. In contrast, computation in the method presented herein is very fast because the features are computed from binary images, and conversion from image features to location vectors involves only one multiplication of matrices.

The method's ability to determine location of part of an object is used to separate the object from its gripper. We found that the technique works as long as at least 40% of the object is visible to the camera. The nonlinearity of the transfer function restricts the range of measurable locations to $\pm$15 mm/8°, with a fairly high accuracy of $\pm$0.1 mm/0.4° (experimental results using a precise x-y-q table). In other words, an error in object grasp can be reduced by a factor of 150 for translation and by a factor of 20 for rotation.

One of the disadvantages of this algorithm is that it depends on robot accuracy, which, in the case of PUMA, is limited to 1 mm/1° (in range of ± 30 mm, ± 15°). The slow segment in the sensor algorithm is the compute-move process, which takes around 5 iterations. The movement of the robot is the slowing factor. In the next section, a technique to reduce the number of iterations is discussed.

B. Future Work

To improve the accuracy of the least-squares estimation module, one could consider an algorithm of constrained least-squares estimation. For example, different weights Wi could be assigned to each feature according to the reliability of each data item. A linearity criterion could be applied: the closer a feature measurement is to the reference feature f0 (linear region), the more reliable are the data, and therefore the higher the assigned weight. We could also use feature standard deviation as a criterion, among other possibilities. Here instead of solving the system $S * \bar{f}' = \bar{P}'$, we have to solve the equation system $W * S * \bar{f}' = W * \bar{P}'$, and the solution for the location vector is:

$$\bar{P}' = [(W*S)^T * (W*S)]^{-1} * (W*S)^T * W*\bar{f}'.$$

To reduce the number of iterations, which would improve sensor speed, one could inquire the partial derivative functions of the image features for each locational coordinate. From these data, one could then construct lookup tables of feature derivatives relative to locations, i.e., a table for each coordinate. Now, instead of using one sensitivity matrix computed at the origin (reference location) we shall use a variety of sensitivity matrices, each dependent object's location. A new sensitivity matrix will be constructed specifically from those lookup tables before each conversion. Using this approach, we shall probably achieve better linear approximation of the transfer function, mapped onto a wider range of object locations. We expect to minimize thereby the number of iterations, while simultaneously extending the measurable locational range.

In the locational application we assume that the shape of an object remains geometrically constant, that any changes in one of its image blobs are attributable to corresponding changes in its location. Another application one can look at is the inspection of indexed objects. In this case, location remains constant, so that any changes in blob features are due to defects in the shape of the inspected object. The sensor will detect a geometric mismatch between a prototype and a similar object.

In the inspection application, the entire object should be visible and the global features should be independent of both position and orientation. A subsequent object to be inspected will be moved to the same location, after which a corresponding feature vector $\bar{f}$ will be measured and computed. A different vector $\bar{f}'=\bar{f}-\bar{f}0$ can then be used to determine whether or not the object is defective. This technique might be applicable for finding tiny defects. Using the sensitivity matrix S, we can obtain a deviation vector $\bar{d} = S^{PI} * \bar{f}'$ that would classify an object as defectives rely on pretraining of defectives.

In general, the mathematic model presented here can be used for location and inspection of an object by processing any kind of data, as long as it presents well and is sensitive to the geometric shape or location of an object. For example, phase delay between transmitter and array of receivers of ultrasound waves reflected from an object, each phase delay measured in a different receiver is a component of a phase delay vector $\bar{f}$. Another example might be to use a 3-D vision system (like the White Scanner); it computes 3-D geometric features and uses an experimental sensitivity matrix to compute an object's location.

References

1. Bolles, R.C., "Robust Feature Matching Through Maximal Cliques,," *SPIE Symposium*, Bellingham, Washington (April 1979).

2. Bolles, R.C., and R.A. Cain, "Recognizing and Locating Partially Visible Objects," *Int. Journal of Robotics Research*, Vol. 1, No. 3 (Fall 1982) 57-82.

3. Gleason, G.J., and G.J. Agin, "A Modular Vision system for Sensor-Controlled Manipulation and Inspection," *9th Int. Symposium and Exposition on Industrial Robots*, pp. 57-70, Washington, D.C. (March 1979).

4. Nahi, N.E., *Estimation Theory and Application*, Krieger Publishing Company, New York, New York, 1976.

5. Paul, R.P., *Robot Manipulators*, MIT Press, Cambridge, Massachusetts, 1981.

6. Smith, R.C., and D. Nitzan, "A Modular Programmable Assembly Station," *Proc. 13th Int. Symposium on Industrial Robots*, pp. 5.53-5.75, Chicago, Illinois (April 18-21, 1983).

7. Strange, G., *Linear Algebra and Its Applications*, Academic Press, Inc., New York, New York, 1980.

Appendix F

ESTIMATING OBJECT LOCATION IN A MANIPULATOR'S HAND USING FORCE/TORQUE INFORMATION

SRI International

ESTIMATING OBJECT LOCATION IN A MANIPULATOR'S HAND USING FORCE/TORQUE INFORMATION

Robotics Laboratory Technical Note

August 1986

By: Aviv Bergman, Research Physicist
Randall C. Smith,* Research Engineer

Robotics Laboratory

The research reported in this paper was supported by the Air Force Office of Scientific Research under Contract F49620-84-K-0007.

SRI Project 7239

---

*Currently with General Motors Research Laboratory, Warren, Michigan.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
415 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Abstract

When an industrial manipulator is commanded to pick up an object, there is some unknown error (difference) between the position and orientation (pose) of the object and that of the manipulator's gripper. In current practice, this error is minimized by using very accurate but costly machines and part fixtures. For many potential applications, expensive fixturing will probably have to be reduced—introducing locational uncertainties that must be minimized by intelligent sensing. An approach is presented here for estimating the pose error of a workpiece in a manipulator's gripper, after the part has been acquired, by sensing wrist force/torque while the manipulator is moving. In principle, it should be possible to estimate the error relationship while the manipulator is transferring the part to its destination, so that the manipulator can correct any errors "on the fly."

1 Introduction

Moving a workpiece from one location to another—whether to place it into a fixture or a machine or to assemble it with another component—is one of the primary functions of an industrial manipulator. In current industrial implementations, these manipulators are surrounded by expensive, special-purpose fixtures and use special-purpose grippers that capture and orient the workpieces with high precision. When the manipulator grasps a workpiece, the part pose in the gripper is assumed to be known approximately by virtue of the positioning accuracy of the manipulator, which is supposed to be sufficiently accurate for the job. Thus, the workcell is preengineered so that a sensor-less robot can perform the task—often at the great expense of designing and building special fixturing. This approach is justified economically for mass production, but for batch manufacturing it will be economically imperative to reduce or eliminate the use of costly fixtures. With no fixtures, however, workpiece locations will become more variable, and when a manipulator grasps a workpiece, the presumed grasp relation and the actual one may be significantly different. Future robotic workcells therefore must accommodate the locational variation efficiently through the use of sensing and intelligence.

If we can determine the pose of a workpiece in a manipulator's gripper, we can estimate the error in the grasp and direct the manipulator to modify its motions to compensate for that error. It is advantageous to determine the pose of the workpiece while it is being held rather than before it is grasped, because without special-purpose fingers and an accurate robot, the act of grasping the workpiece may introduce significant error. Prior work for determining the pose of a part in the hand has been based on vision[1,2], ultrasonics[3], and sparse range or tactile information sensing[4]. In general, sensors to estimate pose error should be mounted on the robot hand; otherwise, it may be necessary to move the held workpiece to a sensing station.

In this paper we present a novel method for determining the pose of a part in a robot hand by using a wrist-mounted force/torque sensor while the manipulator is transfering the part to its destination. Using this method, it should be possible to estimate the grasp error and modify the final manipulator motions to position and orient the workpiece correctly.

2 Overview of Method

In Section 2.1 we describe the coordinate frames used in determining the pose of the object to be grasped. In Section 2.2 we estimate the position (x, y, z) of the center of mass of the grasped object with respect to a coordinate frame of the gripper. This estimation is based on the fact that in an inertial reference frame the torque of a force $\mathbf{F}$ about a given point is given by $\mathbf{r} \times \mathbf{F}$, where $\mathbf{r}$ is the vector from the given point to the point of application of $\mathbf{F}$. In simple terms, if an object is positioned so that its center of mass is directly below the support point, the torque about that point is 0. The torque increases when the horizontal distance is increased. A force/torque sensor, mounted with a known orientation on the manipulator's wrist, will supply the information used in estimating the position, which can then be compared to the forces and torques with the object in the desired grasp pose.

In the second step of the procedure (Section 2.3), we are interested in estimating the orientation error of the held object with respect to the desired orientation in the coordinate frame of the gripper that holds it. From the conservation law of angular momentum and Newton's second law in an inertial reference frame, the torque about a given point is given

by $\mathbf{I}\dot{\omega}$, where $\mathbf{I}$ is the inertia tensor and $\dot{\omega}$ is the angular acceleration about that point. Force/torque sensory information is gathered while the manipulator is moving with a known angular acceleration about that point—again, possibly during the object acquisition or transferral sequence. From the above, the inertia tensor, $\mathbf{I}'$, of the object in its actual grasp position is computed.

Given the inertia tensor, $\mathbf{I}$, in the desired orientation (e.g., from a CAD data base) and the computed inertia tensor $\mathbf{I}'$, the relative orientation, $\mathbf{A}$, can then be computed from the relation $\mathbf{I}' = \mathbf{A}\mathbf{I}\mathbf{A}^T$, where $\mathbf{A}$ is a 3×3 rotation matrix. Because the above relation is nonlinear, we have selected the Newton-Raphson iteration method to compute the elements of $\mathbf{A}$.

2.1 Coordinate Frames

In order to describe the relationships between the coordinate frames, we first establish some notation. The relation of one coordinate frame, F_2, with respect to another frame, F_1, can be described conveniently by a 4×4 homogeneous coordinate transformation matrix [5], which includes a 3×3 upper-left matrix, $\mathbf{R}$, representing the rotation of F_2 with respect to F_1 (about F_1 axes), and a column vector, $\mathbf{p}$, describing the translation of the origin of F_2 from the origin of F_1 (along the axes of F_1). In this paper we denote frame relationships by specifying the rotation and translation separately, i.e., as $(\mathbf{R}, \mathbf{p})$.

The method we use assumes that certain information about the object to be grasped is available from a model data base. We are interested in the mass properties of the object—its mass, center of mass, and inertia tensor. We assume that the object is rigid and that a coordinate Frame, O_1, is attached to it with its origin at the center of mass and its z-axis along the direction of gravity (see Figure 1). In estimating the orientation error, we define a coordinate Frame, O_2, as the frame where the inertia tensor of the object is given. For simplicity and without losing generality, we assume that the model of the inertia tensor of the object is given, defined with respect to the object principal axes. The principal coordinate frame is defined as the frame where the inertia tensor is diagonal. A sensor coordinate Frame, S, is attached to the force and torque sensor. Further, we define a world

coordinate system, W, whose z-axis is parallel to the direction of gravity.

2.2 Estimating the Object Position in the Hand

In the following, we presume that the force/torque sensor mounted on the manipulator's wrist has already been calibrated to remove the effects of the manipulator's hand mass. Description of such calibration is commonly provided with the force sensor. Figure 1 shows the relevant spatial relationships among the force sensor, object, world, and manipulator reference frames. For simplicity, we will estimate the object position with respect to the force sensor frame; the transformation from this frame to the hand frame is assumed to be known.

As described previously, the position estimation is based on the fact that in an inertial reference frame the torque of a force $\mathbf{F}$ about a given point is given by $\mathbf{r} \times \mathbf{F}$, where $\mathbf{r}$ if the vector from the given point to the point of application of $\mathbf{F}$. The position estimation is done while the manipulator is either at rest or moving at a constant velocity; the forces and torques acting upon the object are, therefore, due to gravity only, and the real orientation of the object is not important, so we assign the coordinate O_1, which is aligned with the world coordinate frame W, to the object. Thus, the forces and torques acting upon the object in its coordinate frame 0_1 are:

$$\begin{aligned} \mathbf{F} &= 0\mathbf{i} + 0\mathbf{j} + (mg)\mathbf{k} \\ \mathbf{T} &= 0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k}, \end{aligned} \tag{1}$$

where m is the object's mass (see Figure 2(a)). The force due to gravity is along the z axis of the object coordinate frame with origin at the center of mass, and there are no torques about the object axes. Note that, since the axes of the world frame and frame 0_1 are parallel, the orientation of the force sensor in either frame is $\mathbf{R}^{-1}$ (see Figure 1). The torques acting on the sensor are given by the standard relation [5]

$$\begin{aligned} T_x^S &= \mathbf{n}((\mathbf{F} \times \mathbf{p}) + \mathbf{T}) \\ T_y^S &= \mathbf{o}((\mathbf{F} \times \mathbf{p}) + \mathbf{T}) \\ T_z^S &= \mathbf{a}((\mathbf{F} \times \mathbf{p}) + \mathbf{T}), \end{aligned} \tag{2}$$

where $(\mathbf{n}, \mathbf{o}, \mathbf{a})$ are the components of $\mathbf{R}^{-1}$ along the axes of the force-sensor frame (x_S, y_S, z_S). Note from (2) that the measured torques in the sensor frame are defined in terms of the known force sensor orientation $(\mathbf{n}, \mathbf{o}, \mathbf{a})$, the known object mass, m, and the displacement, $\mathbf{p}$, of the force sensor frame from the object frame. Substituting (1) into (2), we obtain

$$\begin{aligned} T_x^S &= n_x(mg)p_y + n_y(mg)p_x \\ T_y^S &= o_x(mg)p_y + o_y(mg)p_x \\ T_z^S &= a_x(mg)p_y + a_y(mg)p_x. \end{aligned} \tag{3}$$

Knowing $(\mathbf{n}, \mathbf{o}, \mathbf{a})$, m, and the $\mathbf{T}$ components, we can solve (3) for $\mathbf{p} = (p_x, p_y, p_z)$. By aligning the sensor frame and the world coordinates, this solution can be simplified into the following:

$$\begin{aligned} T_x^S &= (mg)p_y \\ T_y^S &= (mg)p_x \\ T_z^S &= 0. \end{aligned} \tag{4}$$

Solutions for p_x and p_y are then easily obtained. To solve for the third displacement, p_z, the object is rotated, for example, by 90° about the y axis. The new torque equations are, then

$$\begin{aligned} T_x^S &= 0 \\ T_y^S &= (mg)p_z \\ T_z^S &= (mg)p_y. \end{aligned} \tag{5}$$

As many different static orientations can be used as desired to overconstrain the results, whose estimates can then be improved through averaging. Thus, the object's displacement relative to the force sensor has been determined. It can be compared with the presumed displacement ($\mathbf{p}$ in Figure 1), so that the robot can correct later motions by the computed difference.

2.3 Estimating the Object Orientation in the Hand

2.3.1 Method of Computation

In this section we are interested in estimating the orientation of the object held by the hand, using information from its force/torque sensor. Again, for simplicity, the orientation will be determined with respect to the force-sensor frame, rather than the gripper frame. From the conservation law of angular momentum and Newton's second law in an inertial reference frame, the torque **T** about a given axis is given by

$$\mathbf{T} = \mathbf{I}\dot{\omega} \tag{6}$$

where **I** is the inertia tensor with respect to the axis of rotation (see Figure 2(b)) and $\dot{\omega}$ is the angular acceleration about that axis. By measuring the torque **T** about the rotational axis, and knowing $\dot{\omega}$, the measured value of **I**, denoted by **I'**, can be computed from (6).

The value of **I** in the desired orientation, denoted by $\mathbf{I}_d$, is derived by using information stored in the CAD data base. Knowing the values of $\mathbf{I}_d$ and **I'**, the difference between the desired orientation and the actual one, represented by rotational error matrix **A**, can thus be computed from the relation

$$\mathbf{I}' = \mathbf{A}\mathbf{I}_d\mathbf{A}^T. \tag{7}$$

In the case where the affect of gravity is included and the sensor coordinate frame is not aligned with the world coordinate frame, i.e., there is an orientational difference, R, between them, the torque measured by the sensor is

$$\mathbf{T} = \mathbf{I}'\mathbf{R}^{-1}\dot{\omega} + (mg)\mathbf{R}^{-1}\mathbf{p}. \tag{8}$$

We now wish to use (8) to solve for **I'**. To simplify this solution, we constrain the z-axis of the sensor frame to be parallel to the z-axis of the world frame, i.e., along the gravitational field. Under this constraint, R is a unity matrix and (8) is reduced to the following:

$$\mathbf{T} = \mathbf{I}'\dot{\omega} + (mg)\mathbf{p}. \tag{9}$$

Substituting $\dot{\omega}_x = \dot{\omega}_y = 0$ and $\dot{\omega}_z = \dot{\omega}$, by inspection of Figure 2(a) we obtain from (9)

$$\begin{pmatrix} T_x^S \\ T_y^S \\ T_z^S \end{pmatrix} = \begin{pmatrix} \dot{\omega} I'_{xz} + (mg)p_y \\ \dot{\omega} I'_{yz} + (mg)p_x \\ \dot{\omega} I'_{zz} \end{pmatrix}. \tag{10}$$

We now use (10) to solve for I'_{xz}, I'_{yz}, and I_{zz}.

To solve for the other components of the inertia tensor, the sensor frame is first rotated by 90 degrees around its x-axis. The old y-axis is now parallel to the world-frame z-axis, and the values of I'_{xy}, I'_{yy}, and I'_{zy} are similarly determined. Finally, I'_{xx}, I'_{xy}, and I'_{xz} components are evaluated in the same way by additional 90-degree rotation of the sensor frame about its current y-axis.

Having determined the elements of $\mathbf{I}'$ and knowing those of $\mathbf{I}_d$, we now use (7) to solve for the elements of the rotational error matrix $\mathbf{A}$. Expanding each element I'_{kl} in (7), where $k, l = 1, 2, 3$, we obtain

$$I'_{kl} = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ki} a_{lj} I_{ij}. \tag{11}$$

One can solve this set of nonlinear equations in a_{ij} by using the Newton-Raphson's iteration method [6],

$$f(\mathbf{a}_n) + (\mathbf{a}_{n+1} - \mathbf{a}_n) \left(\frac{\partial f}{\partial \mathbf{a}}\right)\Big|_{\mathbf{a}_n} = 0, \tag{12}$$

where $\mathbf{a}_n$ is the value of each of the nine a_{ij} elements at the nth iteration, $\mathbf{a}_{n+1}$ is the one at the $(n+1)$th iteration, and $\left(\frac{\partial f}{\partial \mathbf{a}}\right)\Big|_{\mathbf{a}_n}$ is the Jacobian $\mathbf{J}$ at the nth iteration. This process continues until the difference $\Delta \mathbf{a}_n = (\mathbf{a}_{n+1} - \mathbf{a}_n)$ is sufficiently small for all a_{ij} elements. The rotational error may now be corrected by rotating the arm according to the a_{ij} values.

2.3.2 Estimation Accuracy

In order to determine how accurately the estimate will represent the orientation error of the object, we utilize perturbation analysis techniques [6].

The *condition number* of a matrix determines how sensitive the solution of the set of equations is to perturbation. The condition number, $K(\mathbf{J})$, of the Jacobian matrix $\mathbf{J}$ from the last section, is defined as

$$K(\mathbf{J}) = \|\mathbf{J}\|_\infty \|\mathbf{J}^{-1}\|_\infty, \tag{13}$$

where $\|\mathbf{J}\|_\infty$ and $\|\mathbf{J}^{-1}\|_\infty$ are the *norms* of $\mathbf{J}$ and $\mathbf{J}^{-1}$, respectively. From perturbation theory, we can bound the error in estimating the orientation error, $\delta(\Delta\mathbf{a}_n)$:

$$\|\delta(\Delta\mathbf{a}_n)\|_\infty \leq 2U K(\mathbf{J}) \|\Delta\mathbf{a}_n\|_\infty, \tag{14}$$

where U is the resolution of the torque measurement and $\|\Delta\mathbf{a}_n\|_\infty = \max_{1\leq i\leq 9} |\Delta a_i|$.

From (14) one can see that if $K(\mathbf{J})$ is large, then a relatively small perturbation in the measurement will produce a relatively large perturbation in the orientation. In our simulation, the condition number was not greater than 10.0, which is much smaller than a typical condition number characterizing an acceptably behaved system (e.g., 400).

3 Discussion of Results

The method that we describe has not yet been tried experimentally with a real manipulator; however, the computational procedure for determining orientation error of the object has been simulated using a number of different object models and assuming different orientation error magnitudes. The sensor measurement error, U, is included in the simulation, and was taken from the specification of a commercially available force/torque sensor. In this error analysis, however, $\dot{\omega}$ was assumed to be known exactly.

In a particular simulation run, the following parameters were used: the torque sensor resolution, $U = 0.002$ Newton-meters; the (perfectly known) angular acceleration about the direction of gravity (world z axis), $\dot{\omega}_z = 10.0$ *meters per second*^{2}; and an applied orientation error about the object x axis, $\Delta a = 10.0$ *degrees*, with the object presumed to be aligned with the world axes.

The resulting simulation converged in less than 4 iterations to the correct solution with an estimation error of less than 0.1 *degree*.

References

1. Birk, J.R., R.B. Kelley, and V.V. Badami, "Workpiece Orientation Correction With a Robot Arm Using Visual Information," *Proc. 5th International Joint Conference on Artificial Intelligence*, Vol. 1, pp. 758, Massachusetts Institute of Technology, Cambridge, Massachusetts (August 22-25, 1977).

2. Zeiler, E., "Determining an Object's Location in a Robot Hand by Means of Vision," Robotics Laboratory Technical Note, SRI International, Menlo Park, California, August 1984.

3. Tavormina, J.J., and S. Buckley, "Automatic Positioning and Assembly Under Microcomputer Control Via Acoustic Phase Monitoring," *Trans. Society of Manufacturing Engineers*, Vol. 6, pp. 448-454 (1978).

4. Grimson, W.E.L., and T. Lozano-Perez, "Model-Based Recognition and Localization from Sparse Range or Tactile Data," *The International Journal of Robotics Research*, Vol. 3:3, pp. 3-35 (Fall 1984).

5. Paul, R.P., *Robot Manipulators: Mathematics, Programming and Control* (MIT Press, Cambridge, Massachusetts, 1981).

6. Dahlquist, G., and Björck, Å., *Numerical Methods* (Prentice-Hall, Inc., 1974).

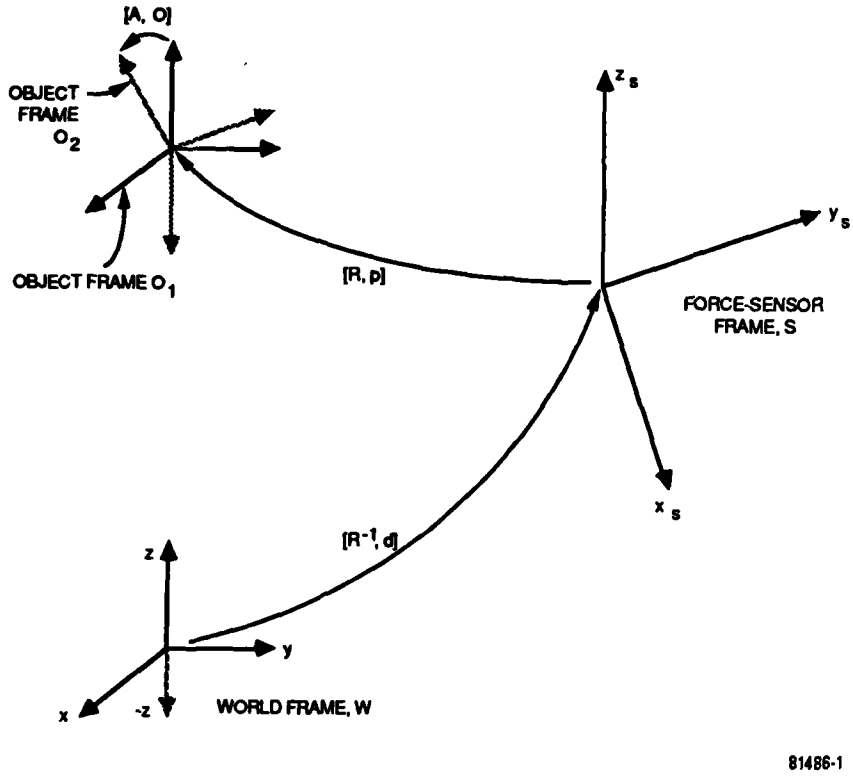
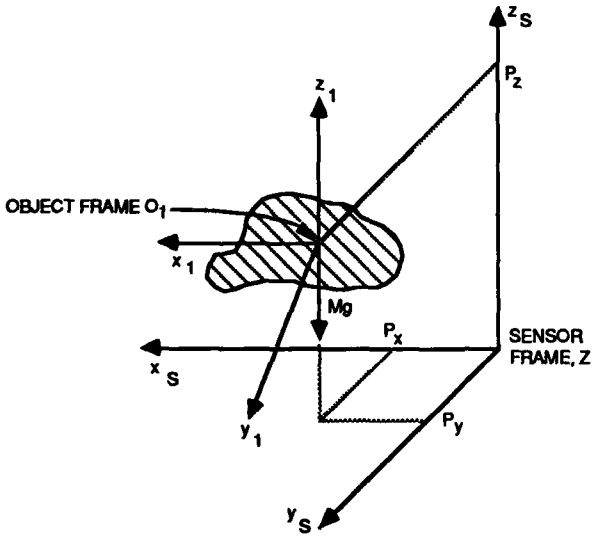
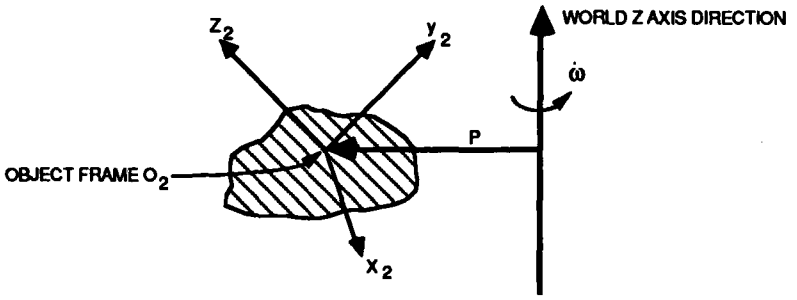


FIGURE 1 COORDINATE FRAMES



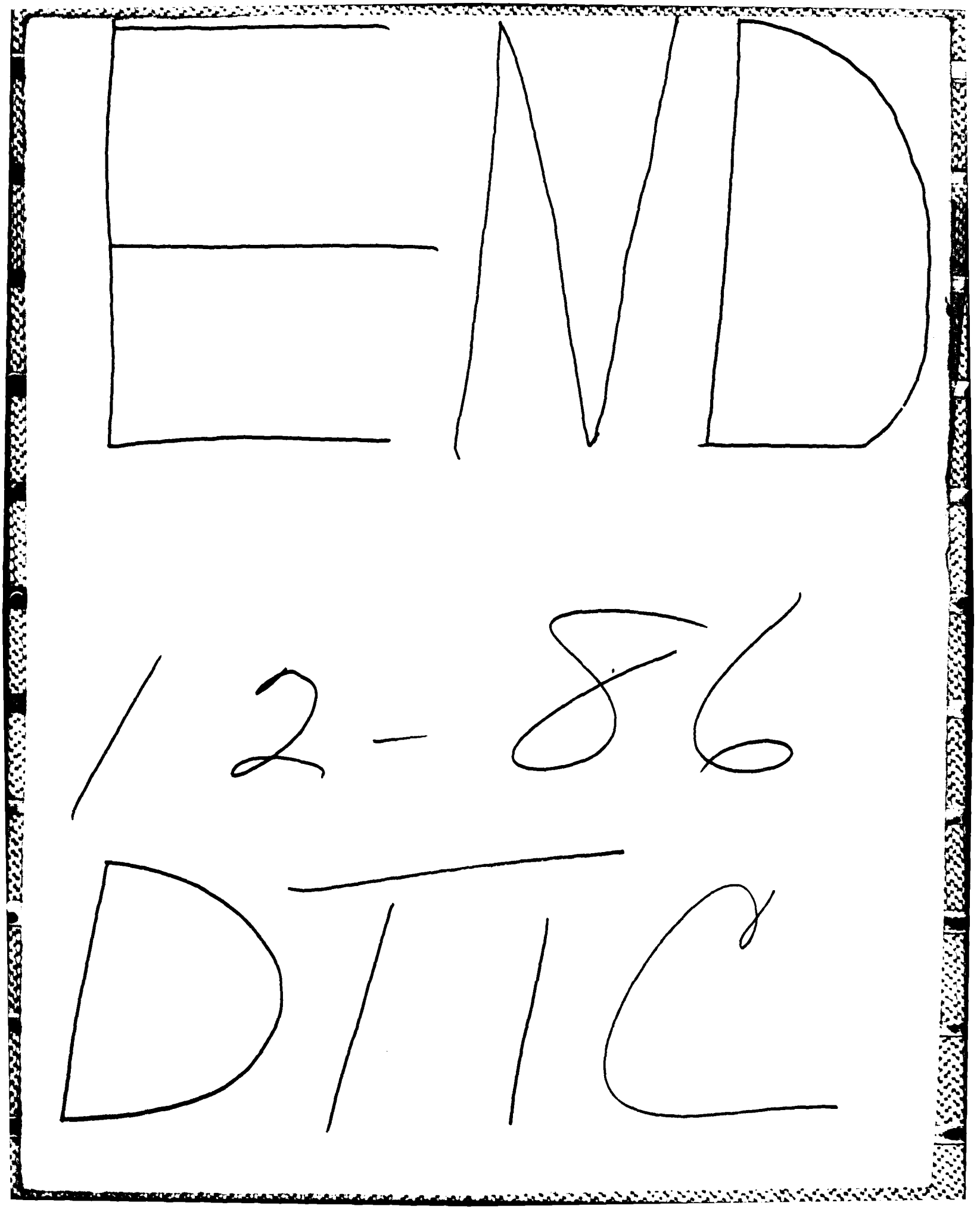
(a) OBJECT POSITION



(b) OBJECT ORIENTATION

81486-2

FIGURE 2 ESTIMATING OBJECT POSITION AND ORIENTATION


END
12-86
DTIC